



Instituto Politécnico Nacional

Escuela Superior de Cómputo



Desarrollo de Sistemas Distribuidos.

Tarea 4. Transferencia de archivos utilizando sockets seguros.

Nombre: Flores Castro Luis Antonio.

Profesor: Pineda Guerrero Carlos.

Grupo: 4CV13.

5 abril 2023

Desarrollo.

Creación de máquina virtual en Ubuntu 18.04 TLS en azure.

Para la creación de la primera máquina virtual, primero debemos seleccionar la opción “Máquinas Virtuales” como se muestra en la figura 1.

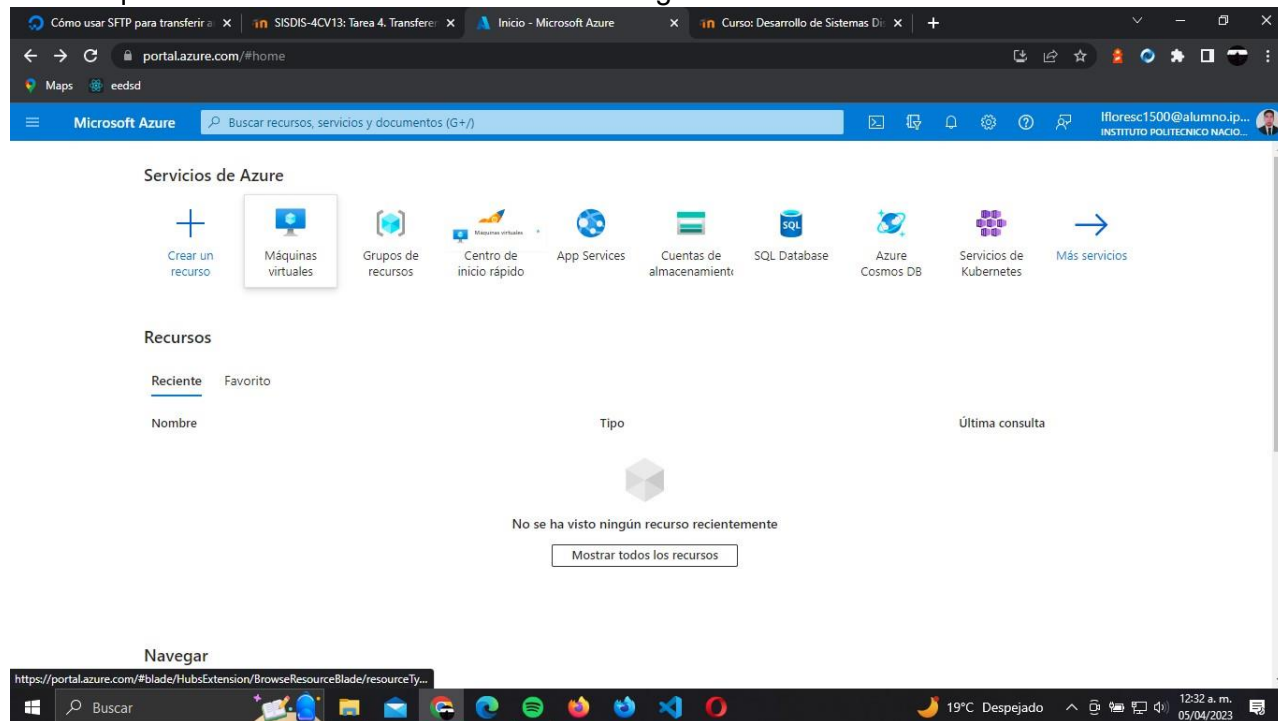


Figura 1. Opción *Máquinas Virtuales*.

Se desplegará una pantalla como se muestra en la figura 1.1, aquí debemos dar click en la opción “Crear”.

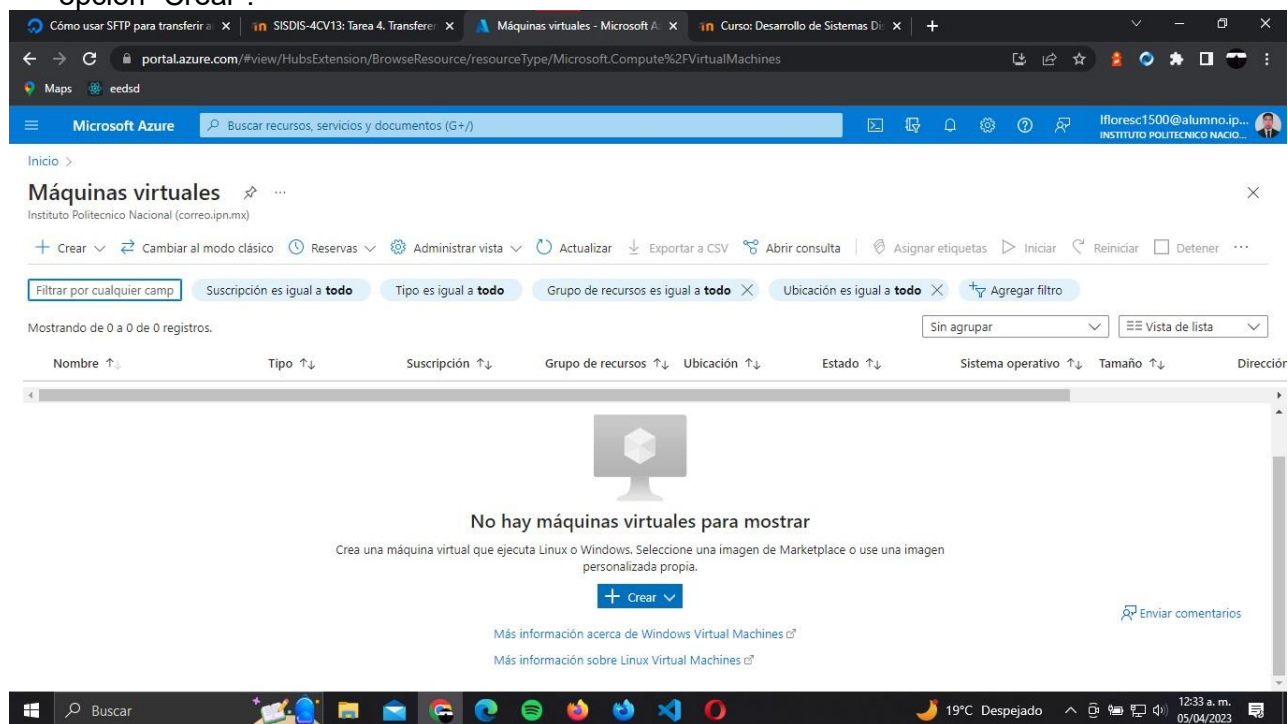


Figura 1.1. Opción “Crear”.

Una vez en la pestaña de datos básicos, se creó un grupo de recursos llamado “Tarea-3”, se asignó el nombre “T4-2019630463” a la máquina virtual y se eligió Ubuntu Server 18.04 LTS como imagen de sistema.

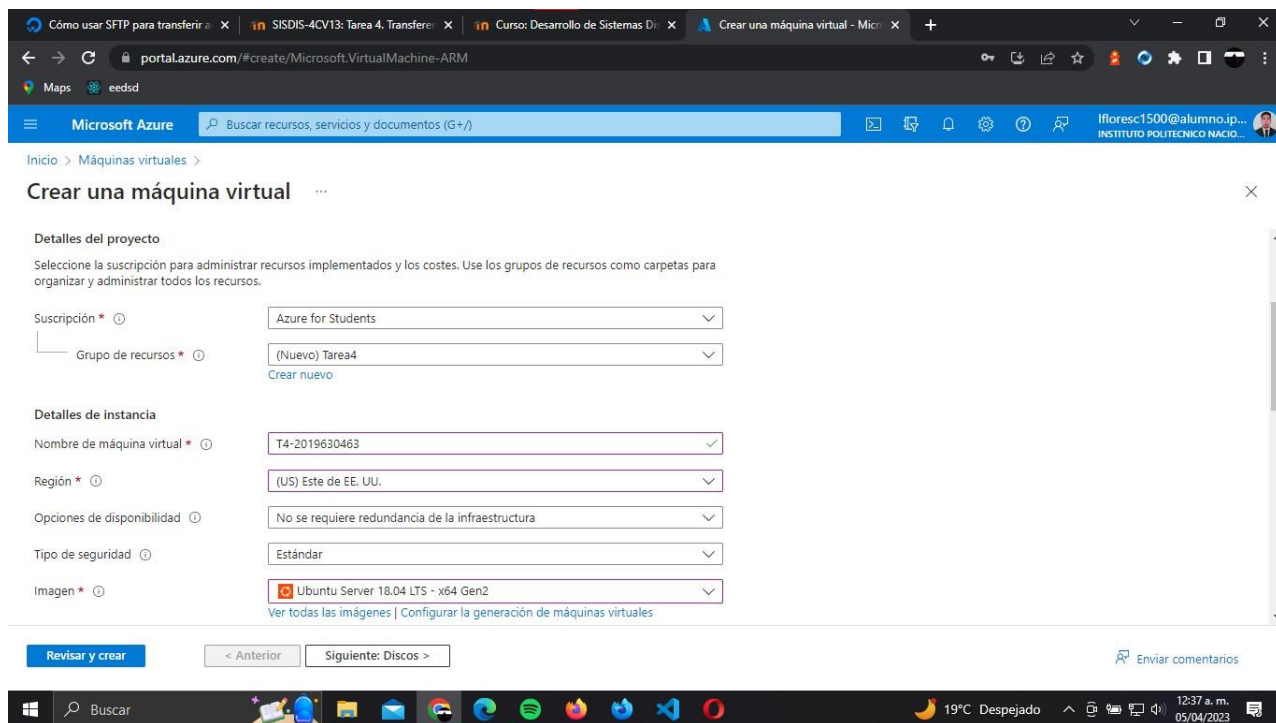


Figura 1.2. Configuración de datos básicos de máquina virtual.

Siguiendo con la configuración de datos básicos, ahora se asigna un nombre de usuario y contraseña, el tamaño de disco se elige de 4GiB para que no haya problema al correr la máquina virtual.

Microsoft Azure portal: Crear una máquina virtual

Tamaño: Standard_B2s - 2 vcpu, 4 GiB de memoria (30,37 US\$/mes)

Cuenta de administrador

Tipo de autenticación: ☐ Clave pública SSH ☒ Contraseña

Nombre de usuario: Servidor

Contraseña: [masked]

Confirmar contraseña: [masked]

Reglas de puerto de entrada

Seleccione los puertos de red de máquina virtual que son accesibles desde la red Internet pública. Puede especificar acceso de red más limitado o granular en la pestaña Red.

Puertos de entrada públicos: ☐ Ninguno ☒ Permitir los puertos seleccionados

Revisar y crear

Figura 1.3. Configuración de datos básicos de máquina virtual.

Seleccionamos la opción permitir los puertos seleccionados para el puerto de entrada SSH (22).

Microsoft Azure portal: Crear una máquina virtual

Contraseña: [masked]

Confirmar contraseña: [masked]

Reglas de puerto de entrada

Seleccione los puertos de red de máquina virtual que son accesibles desde la red Internet pública. Puede especificar acceso de red más limitado o granular en la pestaña Red.

Puertos de entrada públicos: ☐ Ninguno ☒ Permitir los puertos seleccionados

Seleccionar puertos de entrada: SSH (22)

Esto permitirá que todas las direcciones IP accedan a la máquina virtual. Esto solo se recomienda para las pruebas. Use los controles avanzados de la pestaña Redes a fin de crear reglas para limitar el tráfico entrante a las direcciones IP conocidas.

Revisar y crear

Figura 1.4. Configuración de datos básicos de máquina virtual.

Dentro del menú Discos, se selecciona un disco HDD estándar, esto para no generar un gasto mayor en saldo en la cuenta de azure.

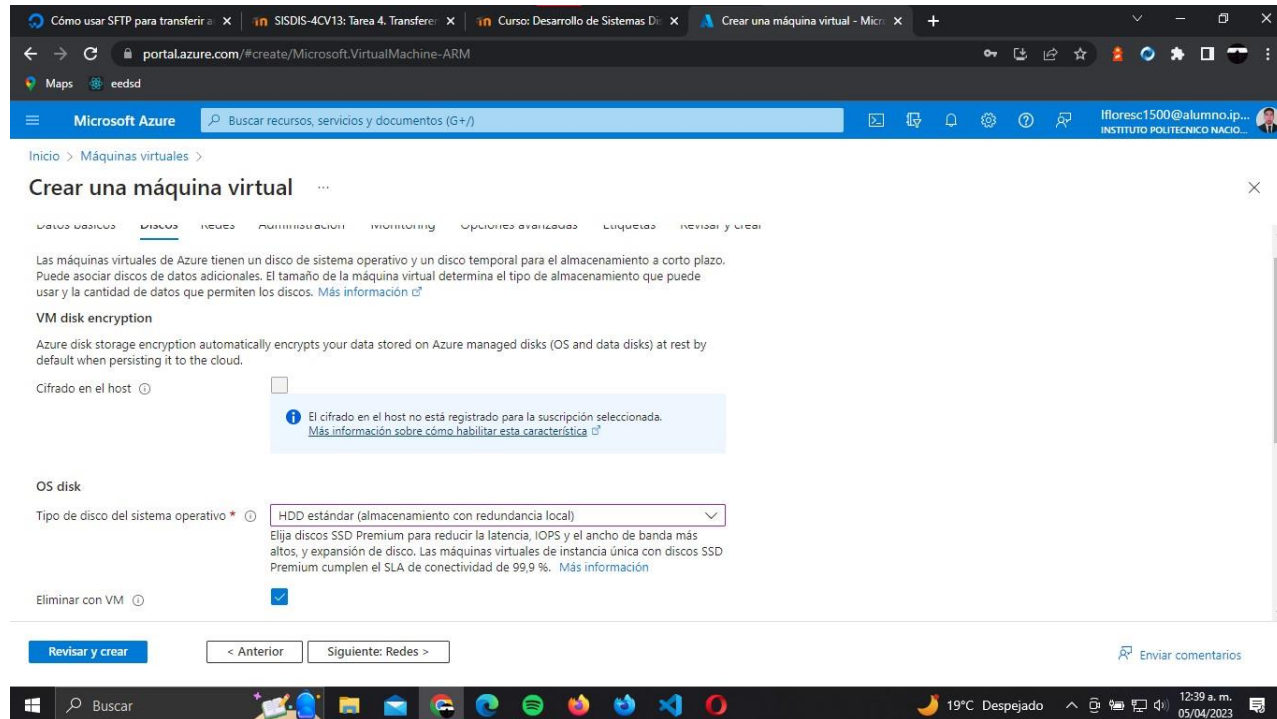


Figura 1.5. Configuración de discos de máquina virtual.

Pasando al menú de “Redes”, dejamos la configuración como se encuentra por default.

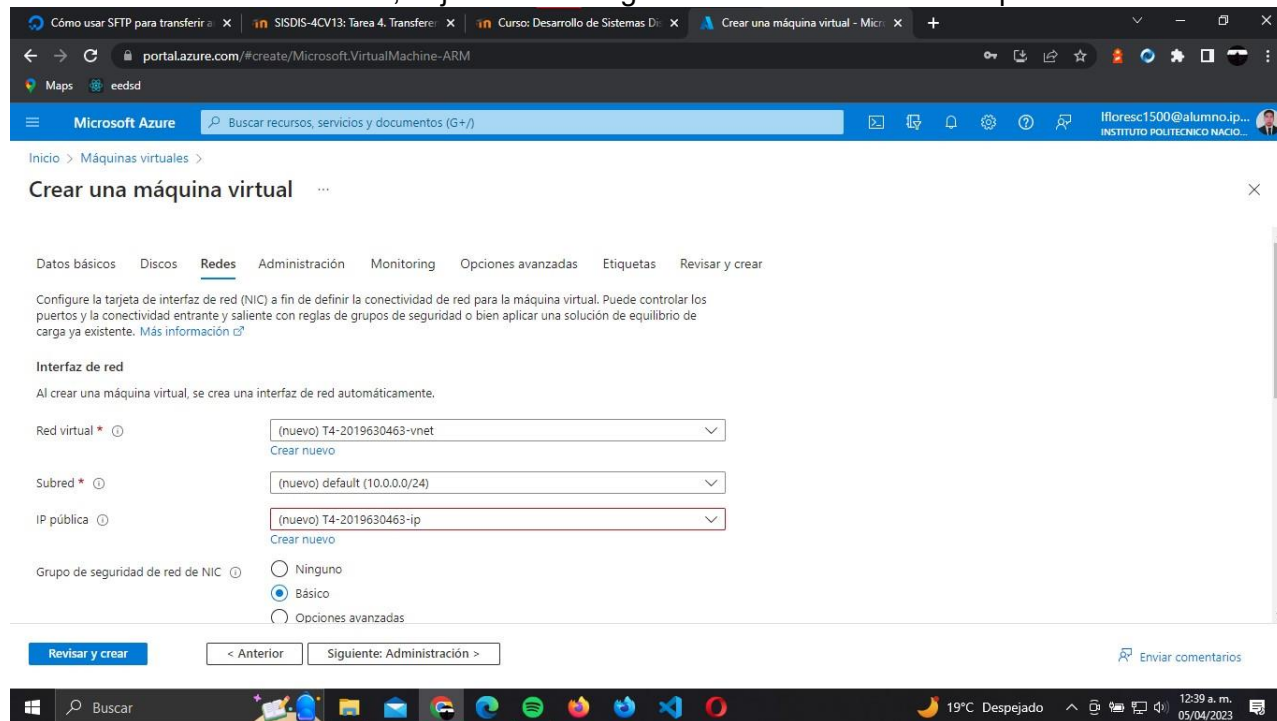


Figura 1.6. Configuración de redes de máquina virtual.

En la parte de “Administración” de igual forma, mantenemos la configuración inicial.

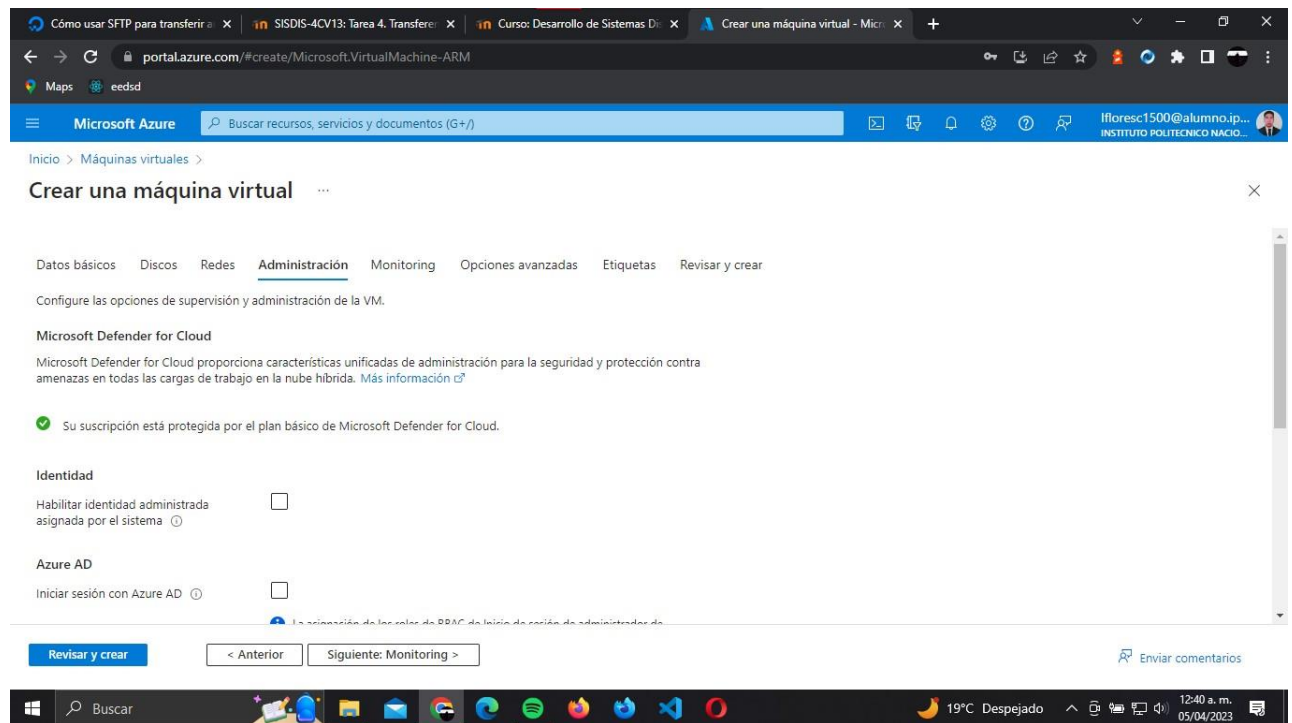


Figura 1.7. Configuración administración de máquina virtual.

En el menú “Monitoring” conservamos la configuración que se presenta por default.

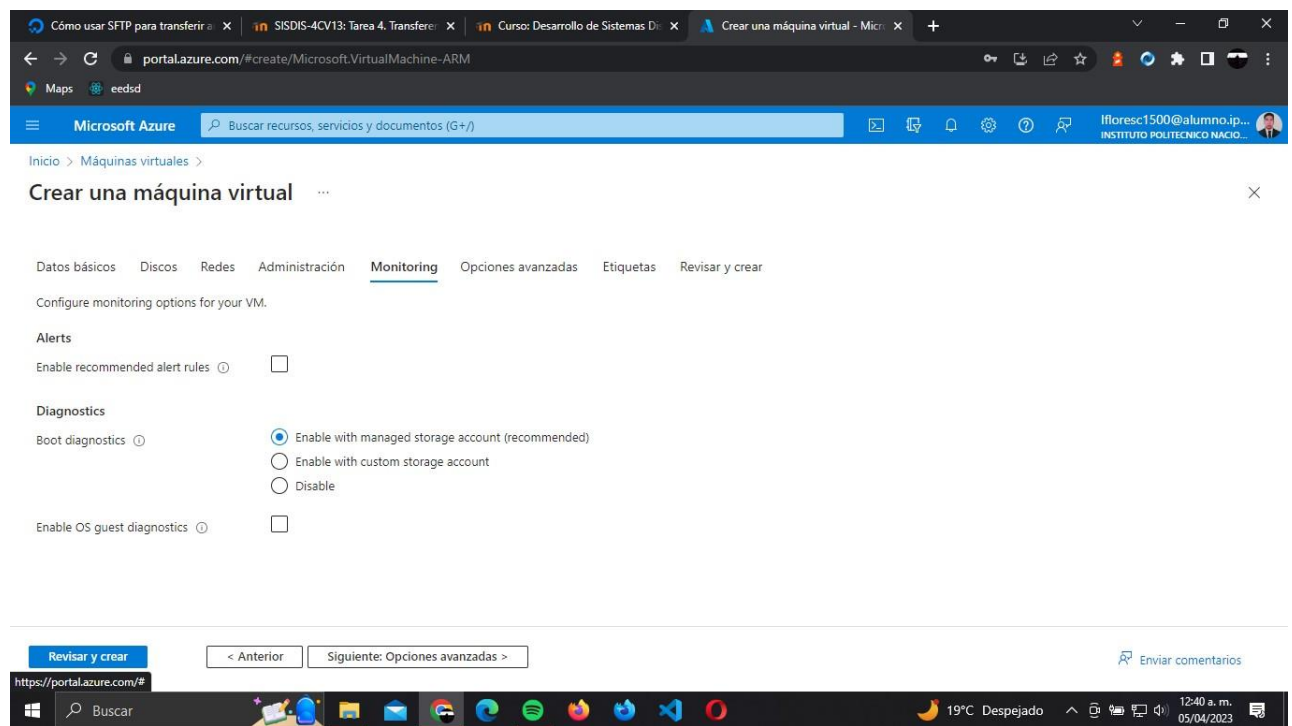


Figura 1.8. Configuración monitoring de máquina virtual.

En la parte de “Opciones avanzadas” de igual manera se conserva la configuración por default.

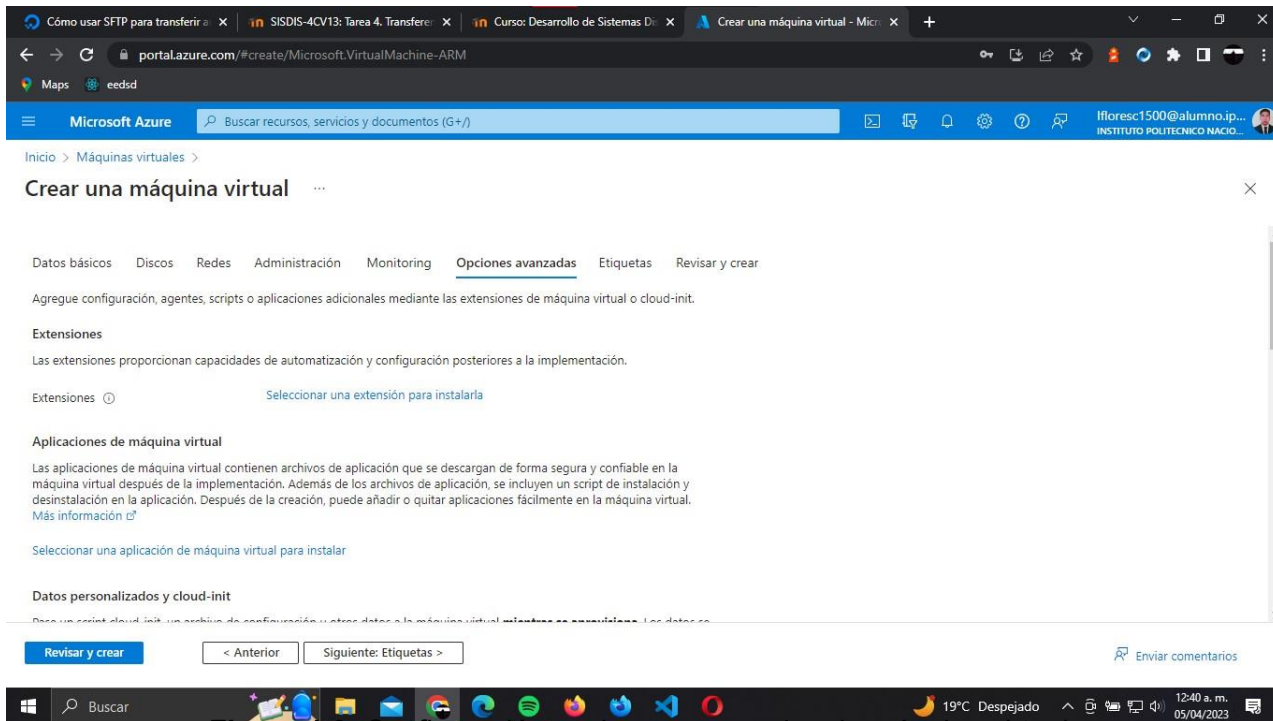


Figura 1.9. Configuración opciones avanzadas de máquina virtual.

En el menú “Etiquetas” dejamos la configuración por default tal y como esta.

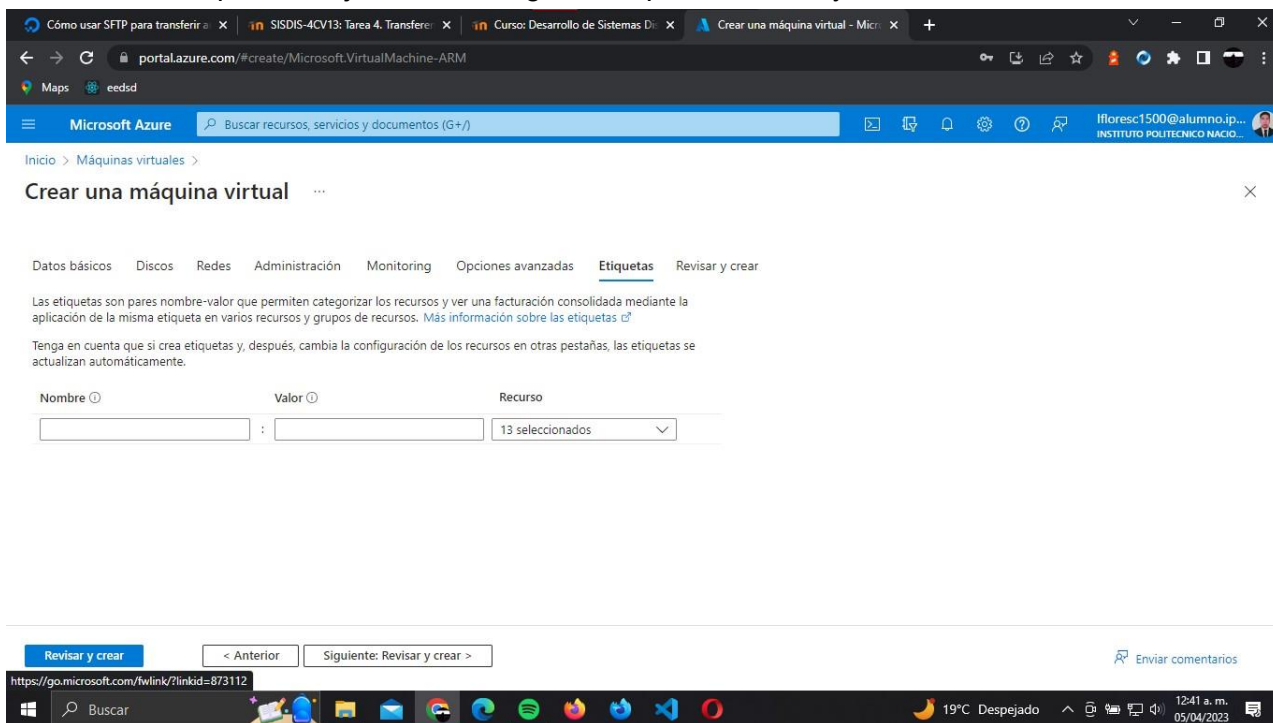


Figura 1.10. Configuración etiquetas de máquina virtual.

Una vez configurado todo, ahora dentro de revisar y crear, nos aparece el resumen de configuración de la máquina virtual, así que se procede a crearla.

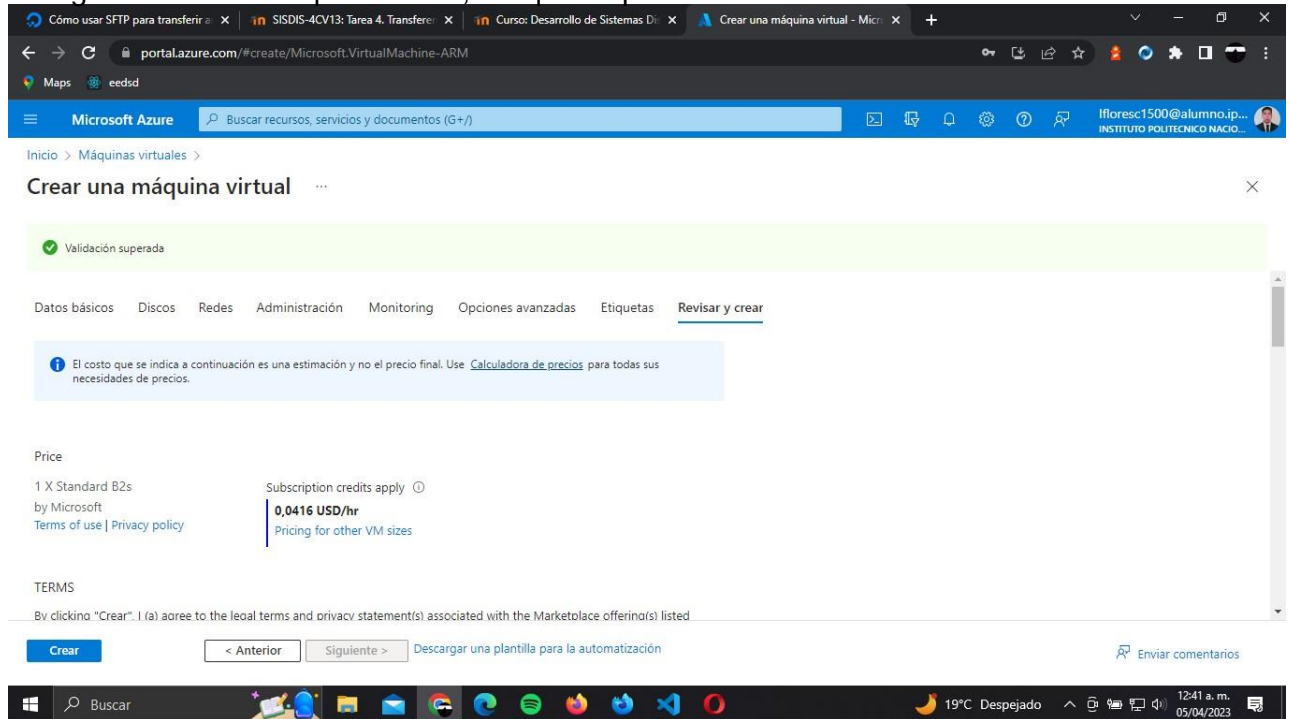


Figura 1.11. Resumen de configuración de máquina virtual.

En la figura 1.12 se muestra que la creación de la máquina virtual este en proceso.

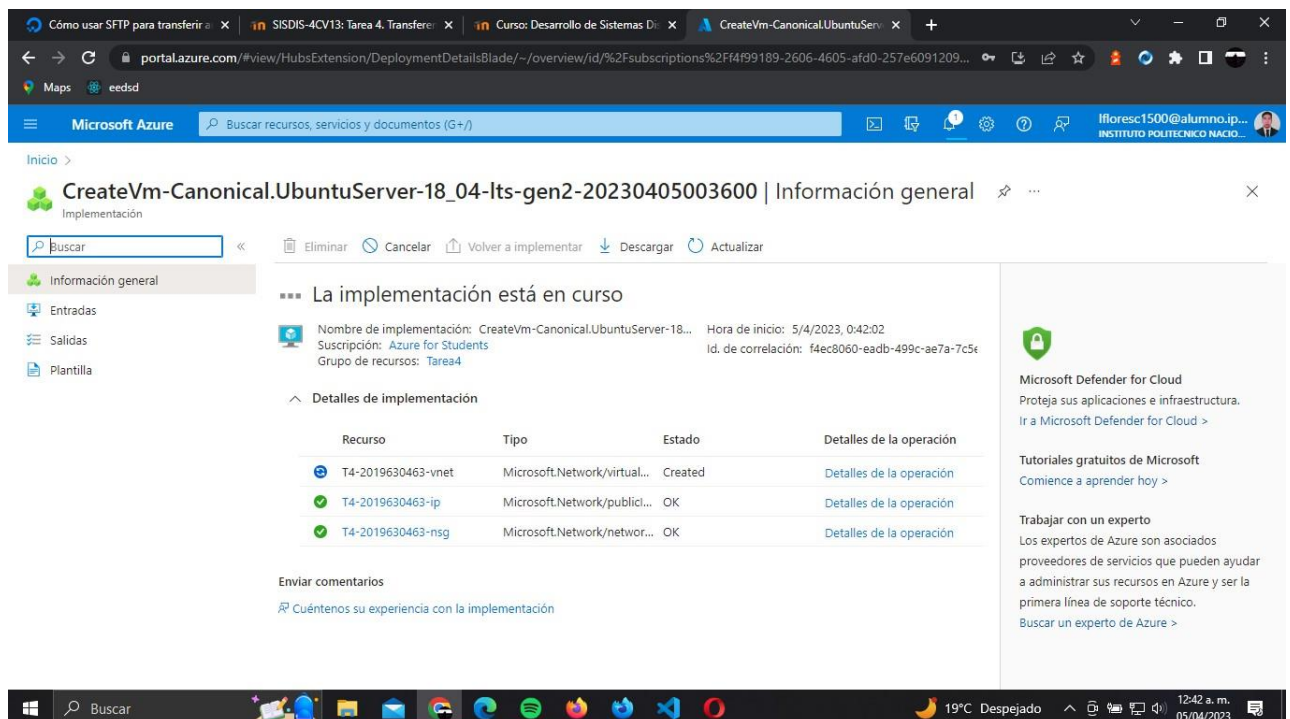


Figura 1.12. Resumen de configuración de máquina virtual.

En la figura 1.13, se muestra que la máquina virtual ha sido creada de forma exitosa.

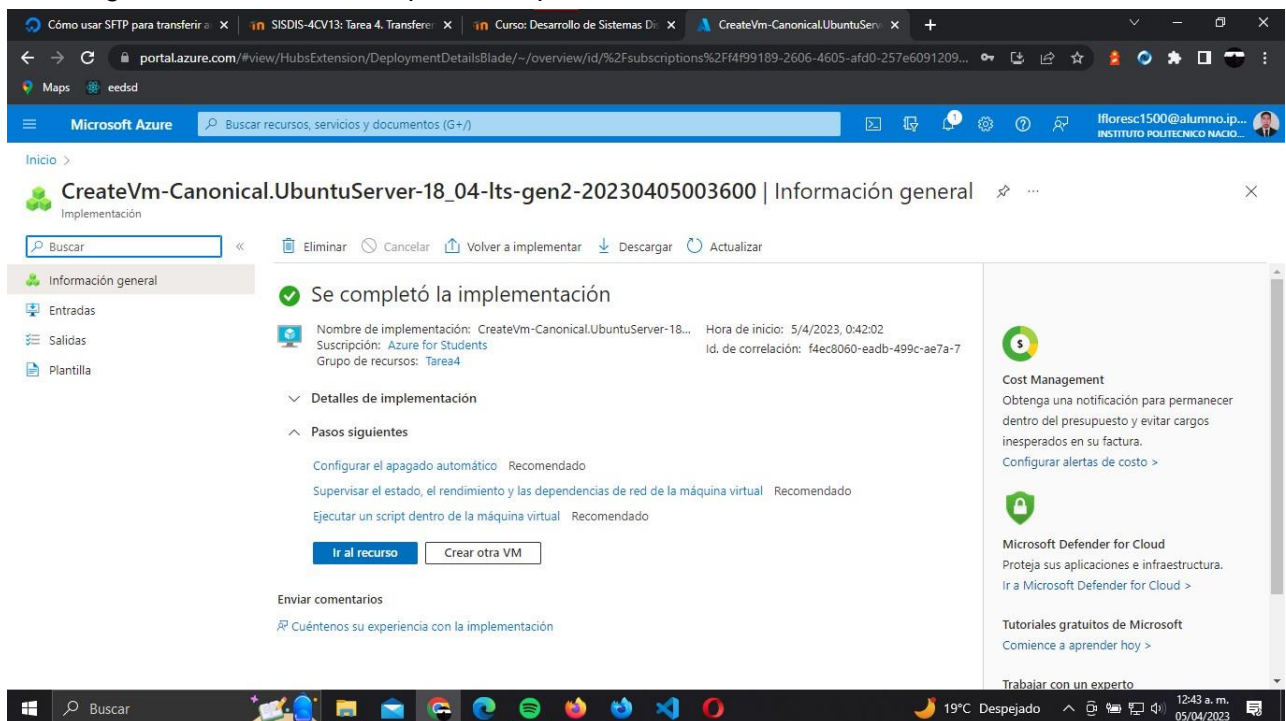


Figura 1.13. Creación de máquina virtual finalizada.

Se despliega la información general de la máquina virtual, donde se muestra el nombre, dirección IP pública, entre otra información de esta.

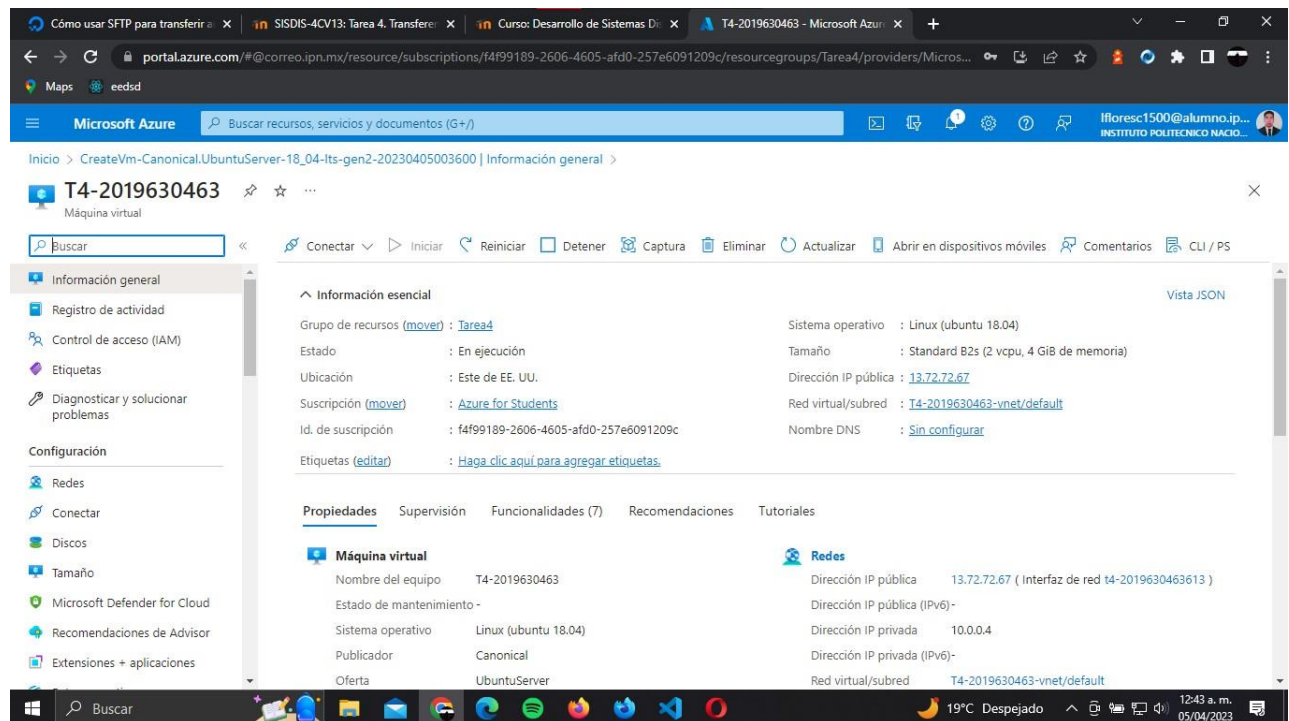


Figura 1.14. Información general de la máquina virtual.

Ahora se realiza la configuración del puerto 8000, ya que este será utilizado para la ejecución de los programas.

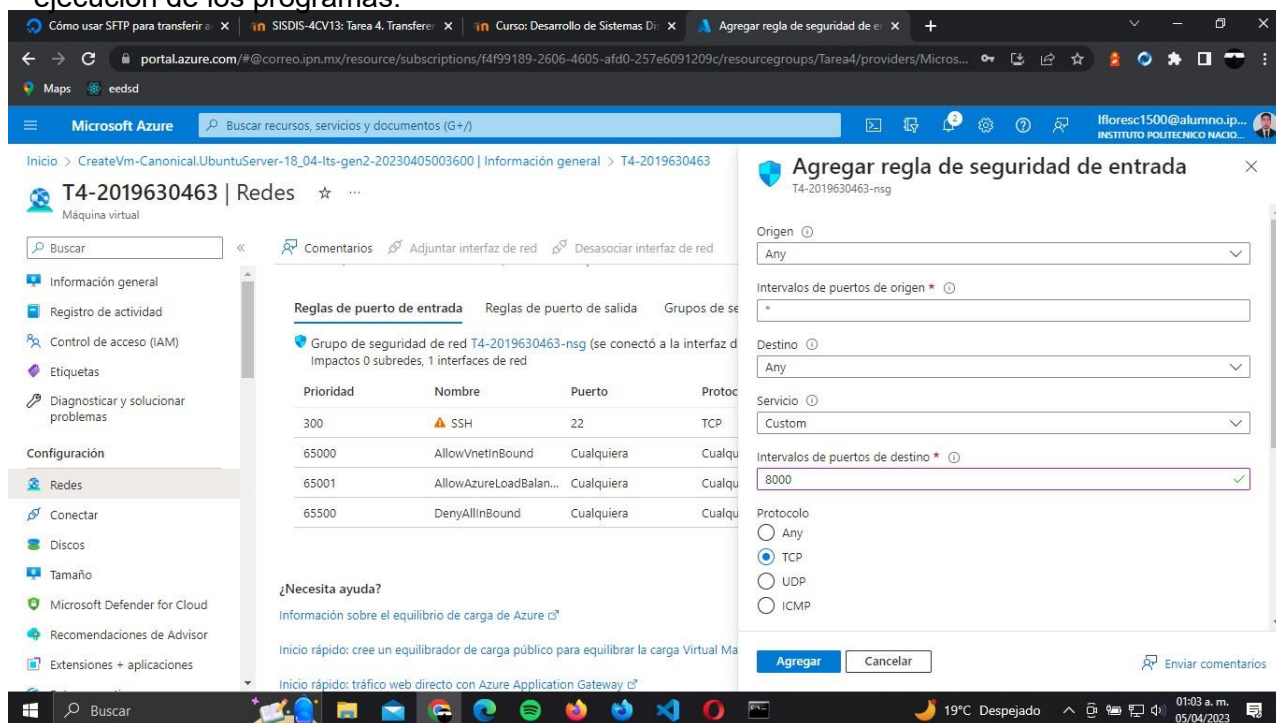


Figura 1.15. Configuración de puerto 8000 en máquina virtual.

Ya bien configurada la máquina virtual ahora se realiza la conexión con ssh Servidor@13.72.72.67 desde Windows.

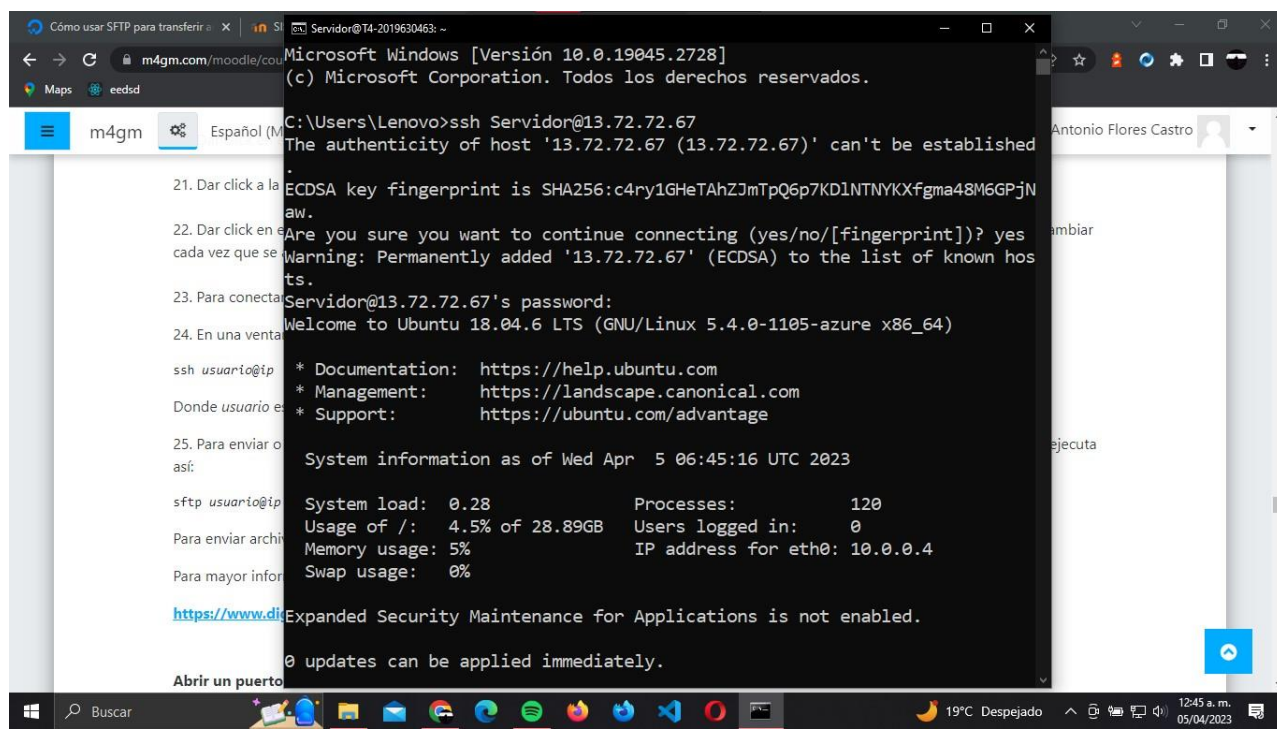


Figura 2. Conexión a máquina virtual mediante ssh en terminal de windows.

Mediante sftp realizamos el envío del Servidor.java a la máquina virtual.

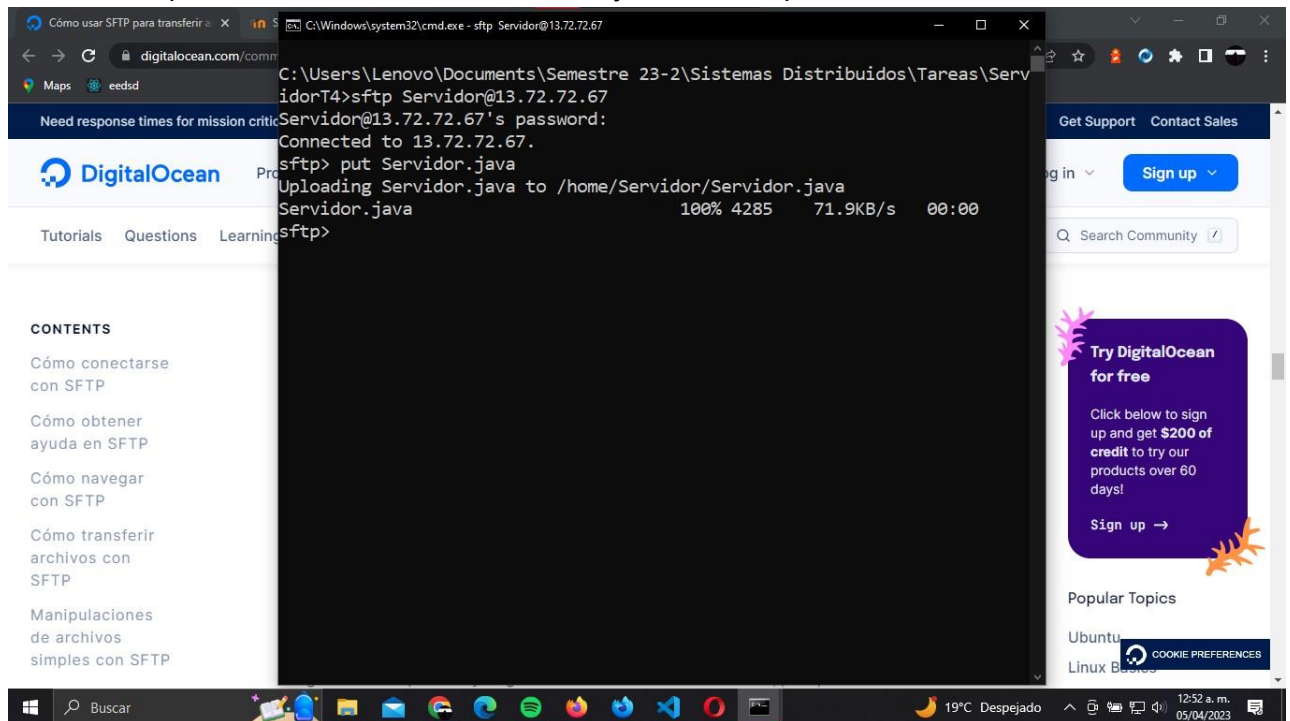


Figura 2.1 Transferencia de archivos mediante sftp a la máquina virtual.

Ya dentro de la máquina virtual ahora creamos el repositorio de claves y certificados de seguridad.

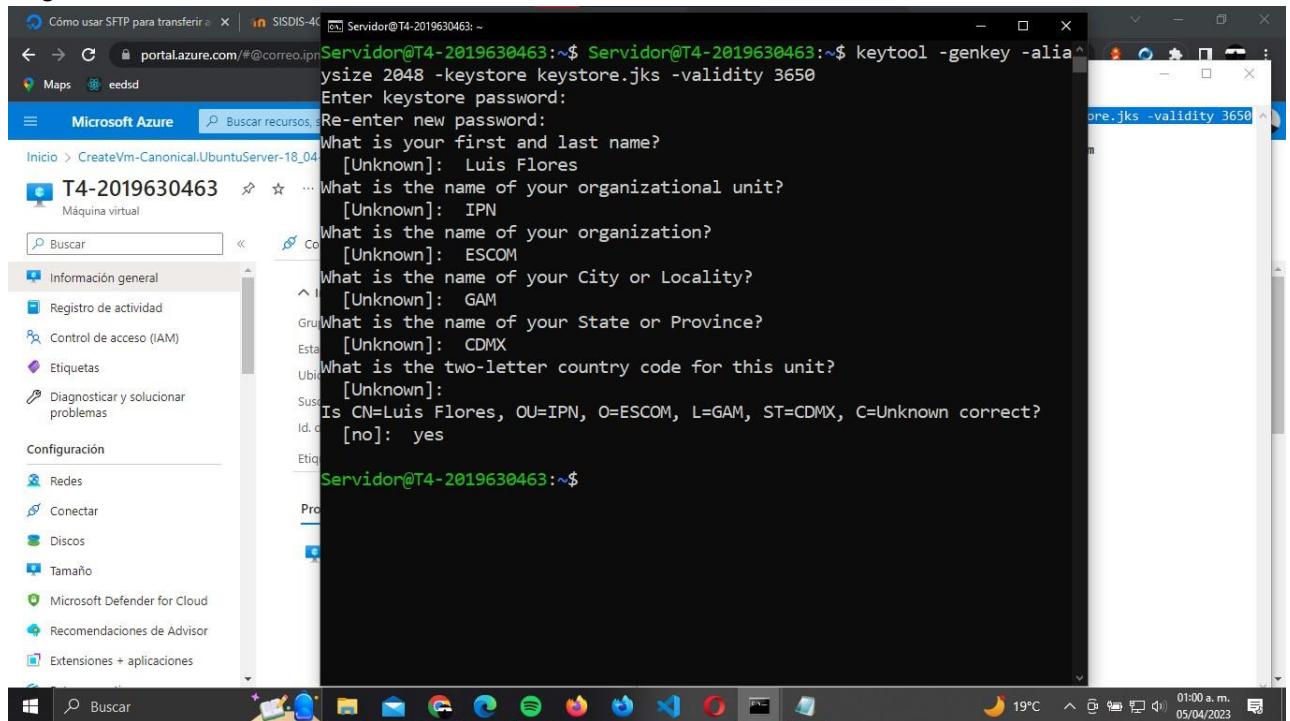


Figura 2.2 Creación de repositorio de claves y certificados de seguridad.

Una vez creado el repositorio de claves y certificados ahora creamos el certificado “.pem”, esto de igual forma dentro de la máquina virtual.

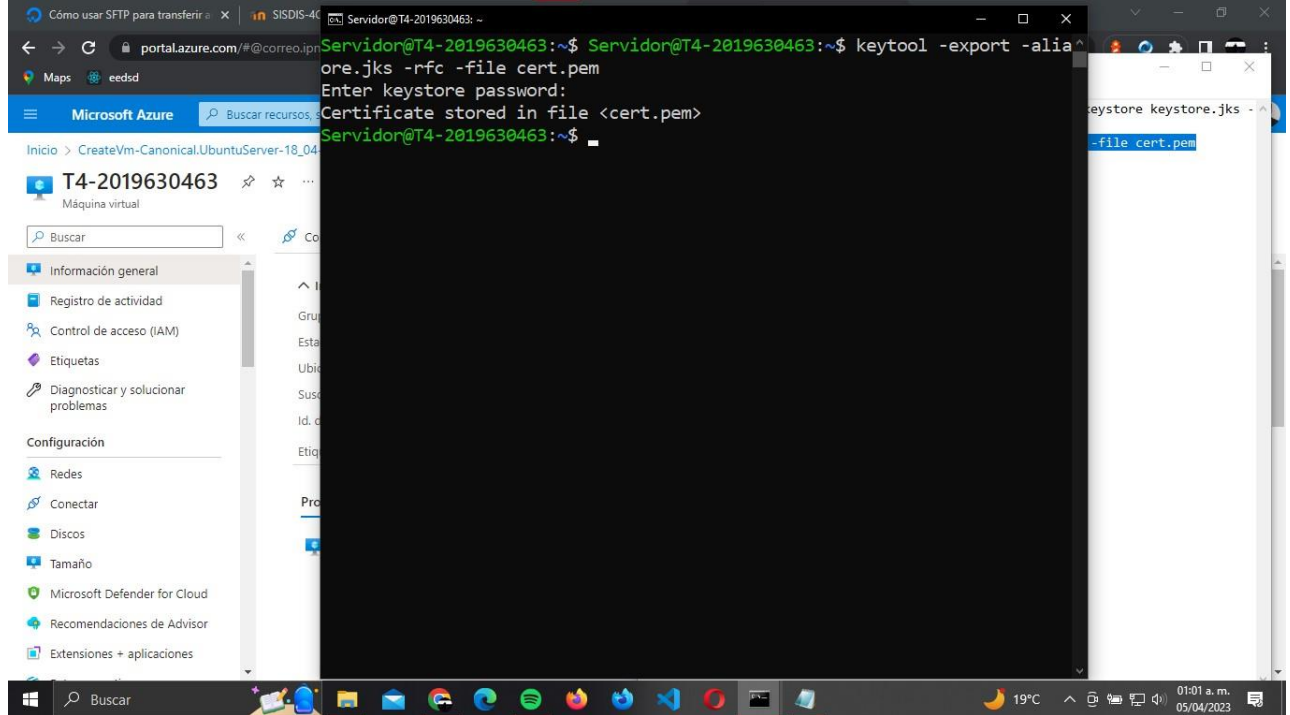


Figura 2.3 Creación del certificado .pem.

Ya creados tanto el repositorio y el certificado ahora los descargamos con sftp, para que la compilación y ejecución de los clientes PUT y GET se realicen de manera correcta.

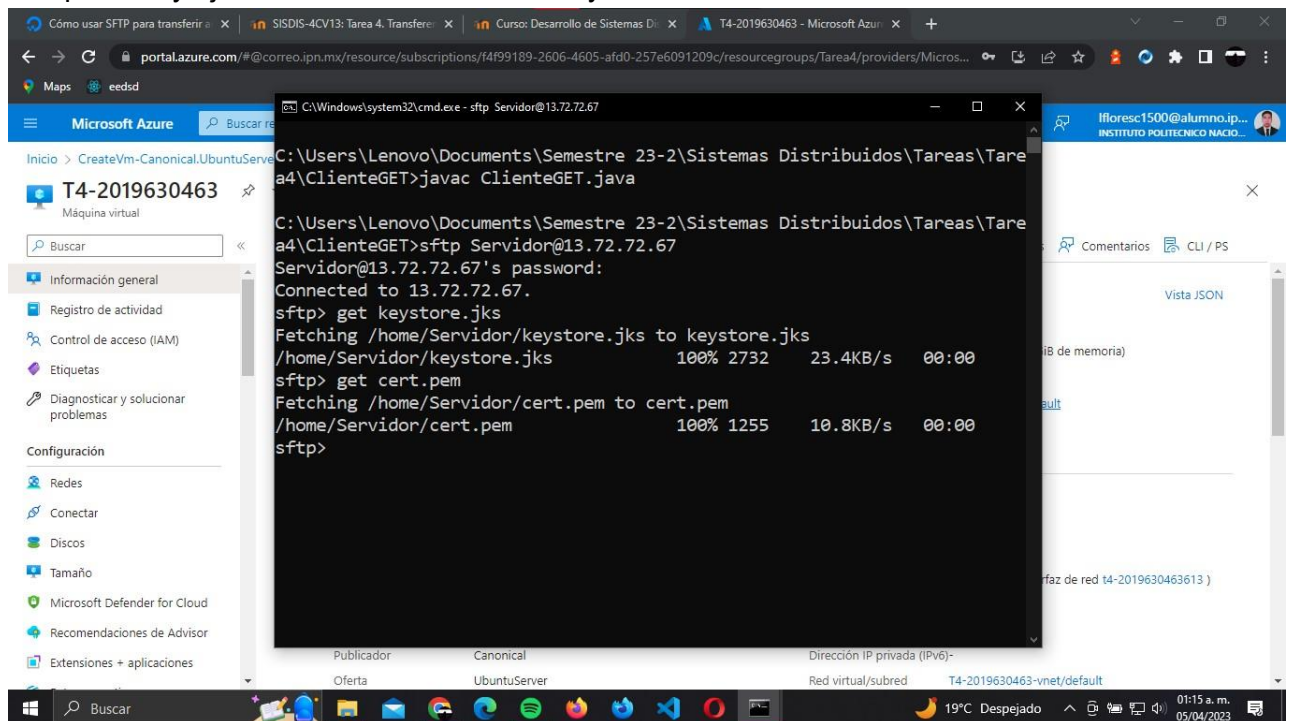


Figura 2.4 Transferencia mediante sftp de repositorio y certificado a máquina local.

Ahora se compila y ejecuta el Servidor en la máquina virtual, se muestra un mensaje donde indica que esta esperando por una conexión.

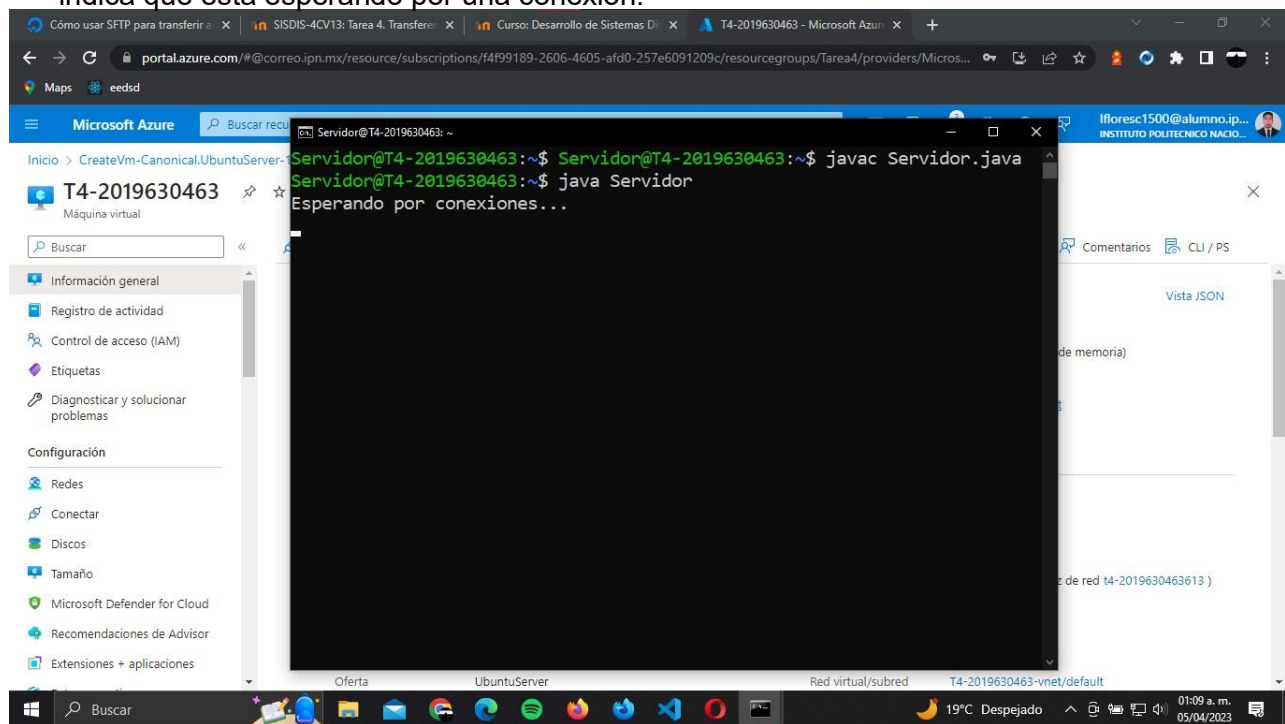


Figura 2.5 Compilación y ejecución del Servidor.java

En la figura 2.6 se muestra la carpeta donde se encuentra el ClientePUT.java y el archivo "Promt.txt" el cual será enviado al servidor.

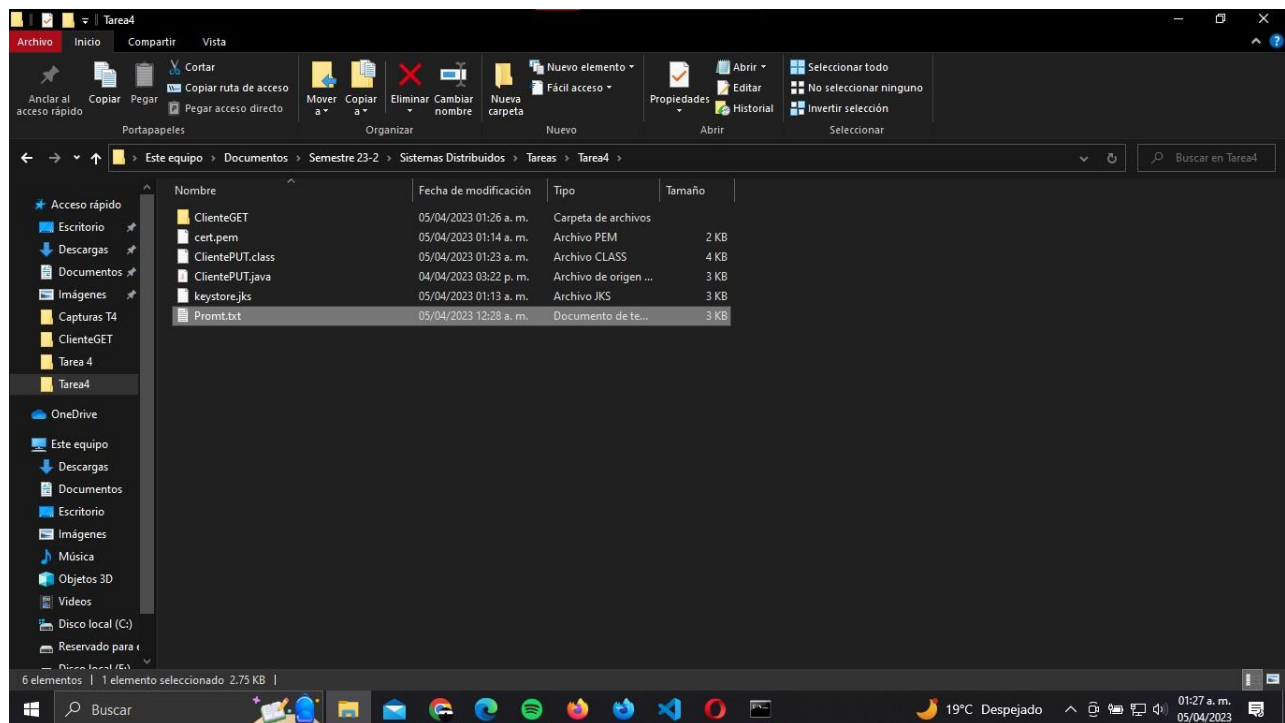


Figura 2.6 Ubicación de archivo ClientePUT.java y archivo "Promt.txt".

En la figura 2.7 se muestra el contenido del archivo “Promt.txt”.

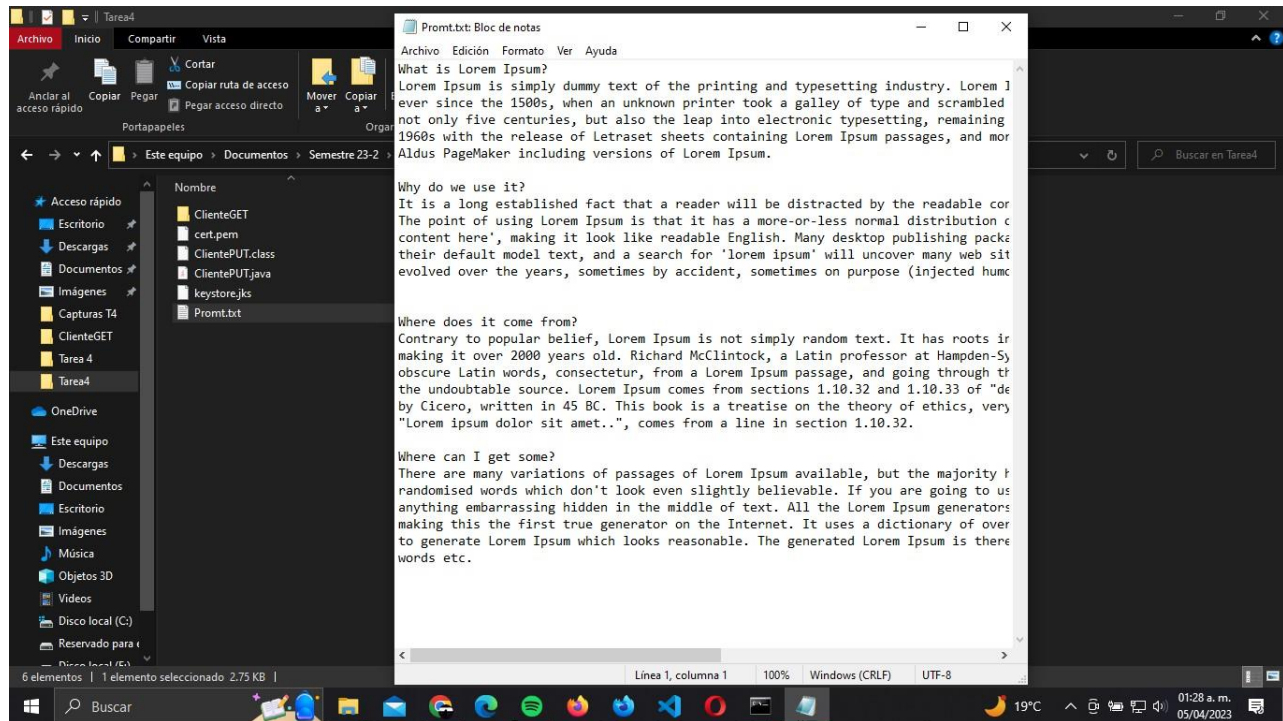


Figura 2.7 Contenido del archivo “Promt.txt”.

El clienteGET.java ha sido guardado en otra carpeta para verificar que efectivamente se esta realizando la escritura del archivo.txt, en la figura 2.8 se muestra que el archivo “Promt.txt” aun no ha sido escrito ya que aun no se ha ejecutado el programa.

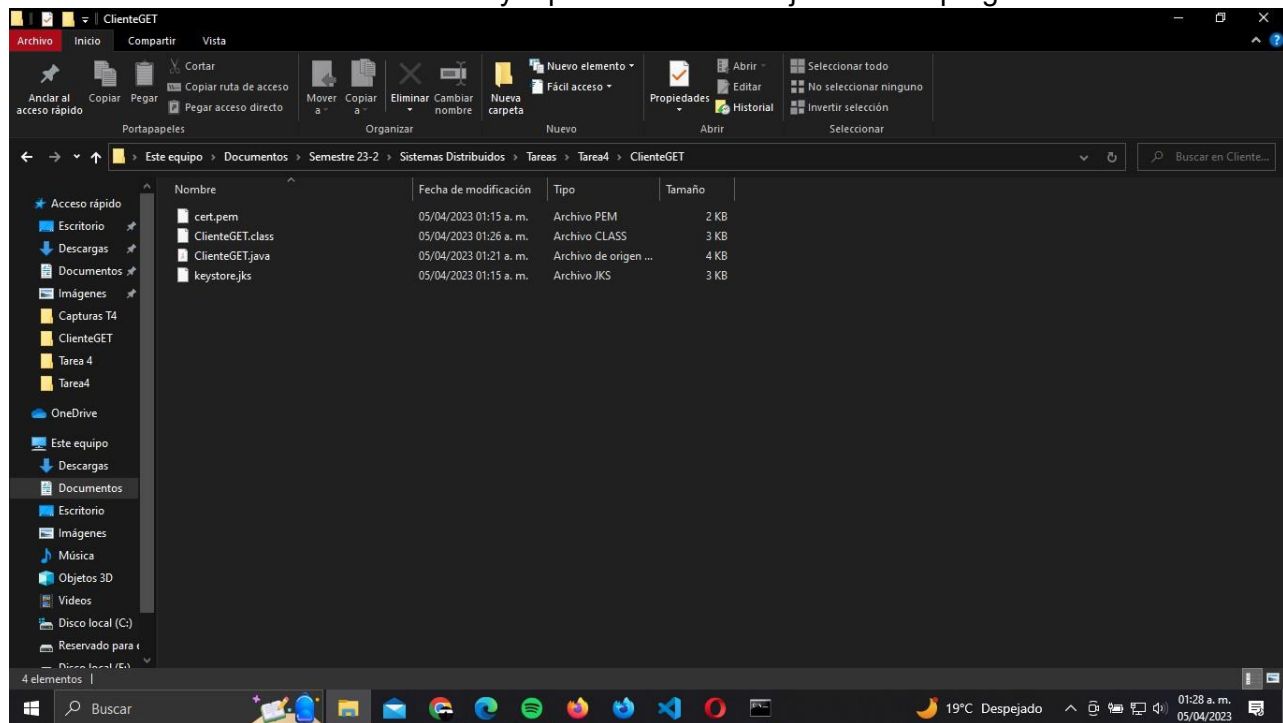


Figura 2.8 Ubicación del programa ClienteGET.java

Ahora se realiza la compilación y ejecución del ClientePUT.java, se la pasa como parámetros al programa la IP del servidor en este caso 13.72.72.67, el número de puerto a utilizar, puerto 8000 y el nombre del archivo a mandar al servidor "Promt.txt", se puede ver que se despliega un mensaje por parte del servidor donde se indica el archivo fue recibido por el servidor exitosamente.

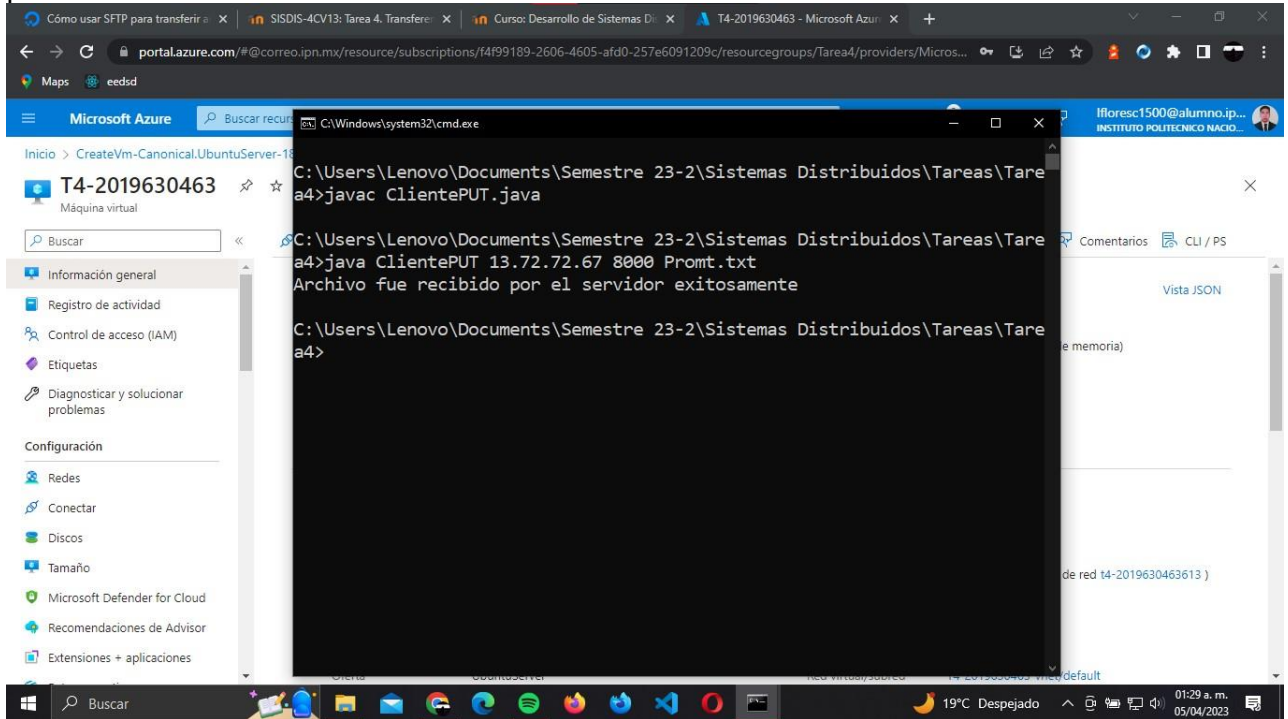


Figura 2.9. Compilación y ejecución del ClientePUT.java

Del lado del servidor podemos ver que la conexión del ClientePUT fue aceptada.

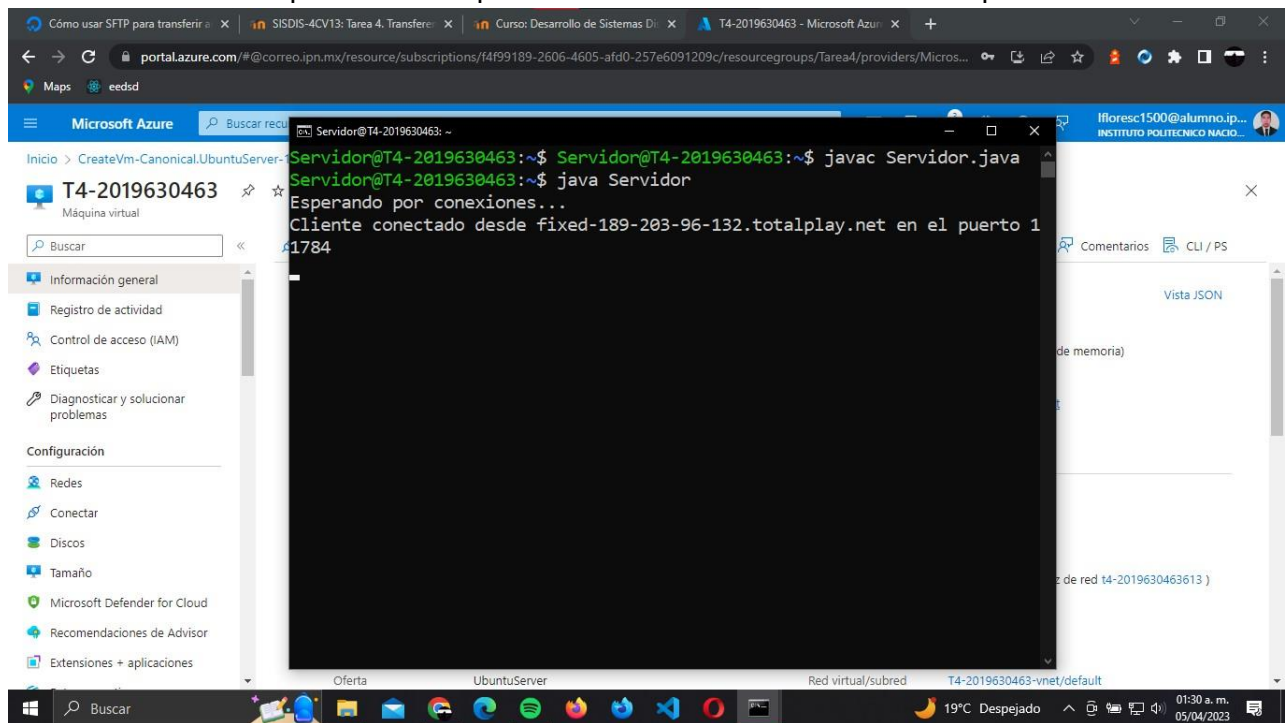


Figura 2.10. Conexión del ClientePUT aceptada por el servidor.

Después procedemos a compilar y ejecutar el ClienteGET.java, a este programa se la pasa como parámetros la IP del servidor 13.72.72.67 el puerto 8000 y el nombre del archivo a escribir "Promt.txt", se puede ver que el programa muestra un mensaje dónde se indica que el archivo fue recibido exitosamente.

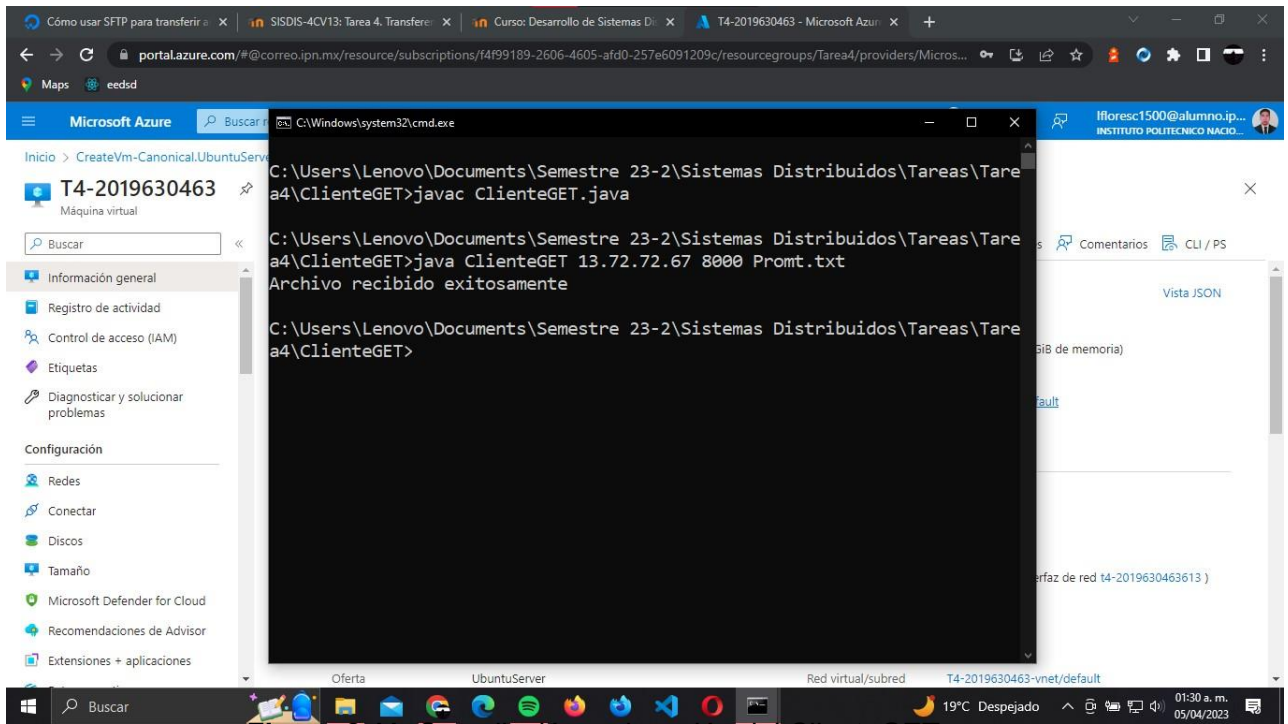


Figura 2.11. Compilación y ejecución del ClienteGET.

Del lado del servidor podemos ver que de igual forma la conexión del ClienteGET fue aceptada.

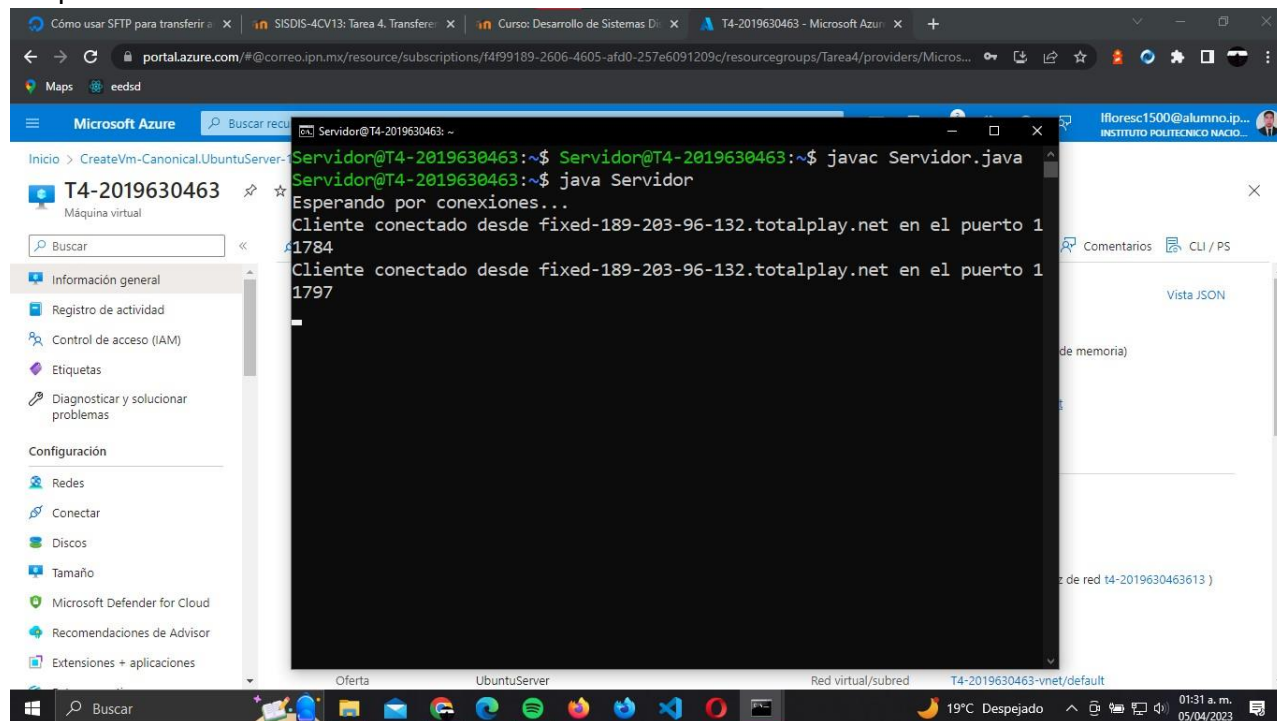


Figura 2.12. Conexión del ClienteGET aceptada por el servidor.

Dentro de la ubicación del ClienteGET se puede ver que se realizó la escritura del archivo “Promt.txt”.

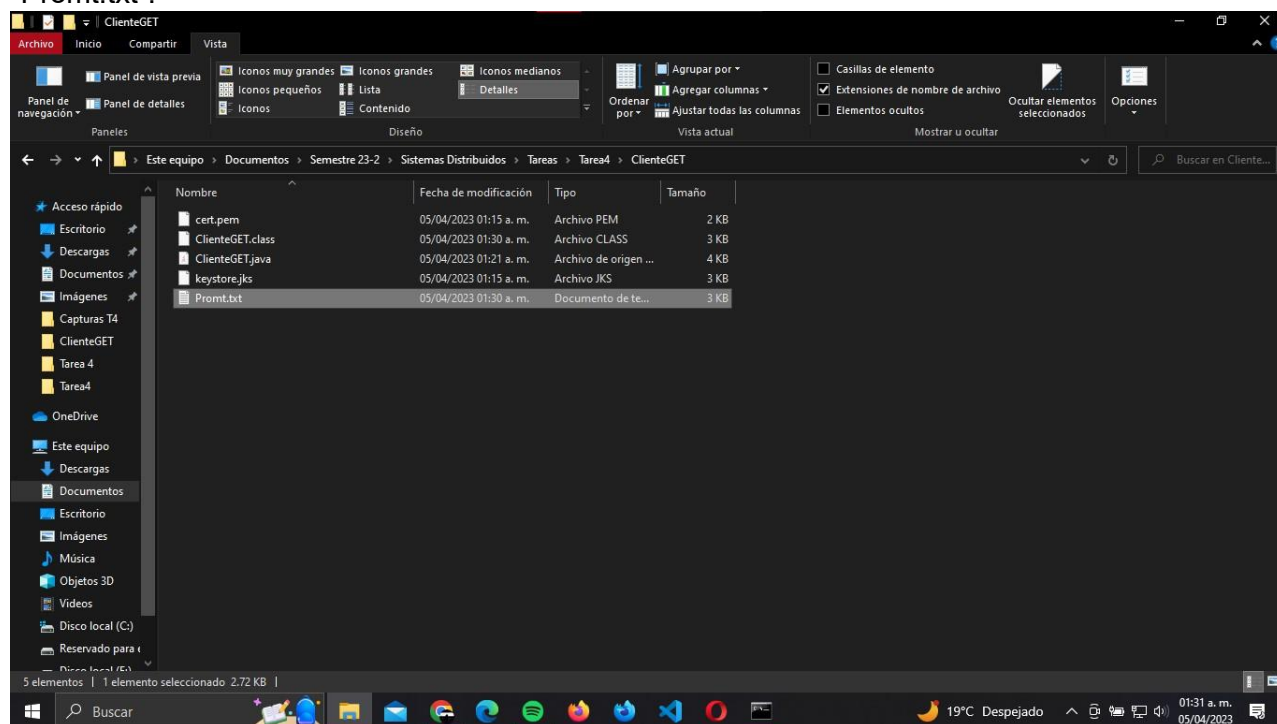


Figura 2.13. Escritura del archivo “Promt.txt” en la ubicación del ClienteGET.

Abrimos el archivo “Promt.txt” para verificar que se haya realizado la escritura del contenido de forma correcta.

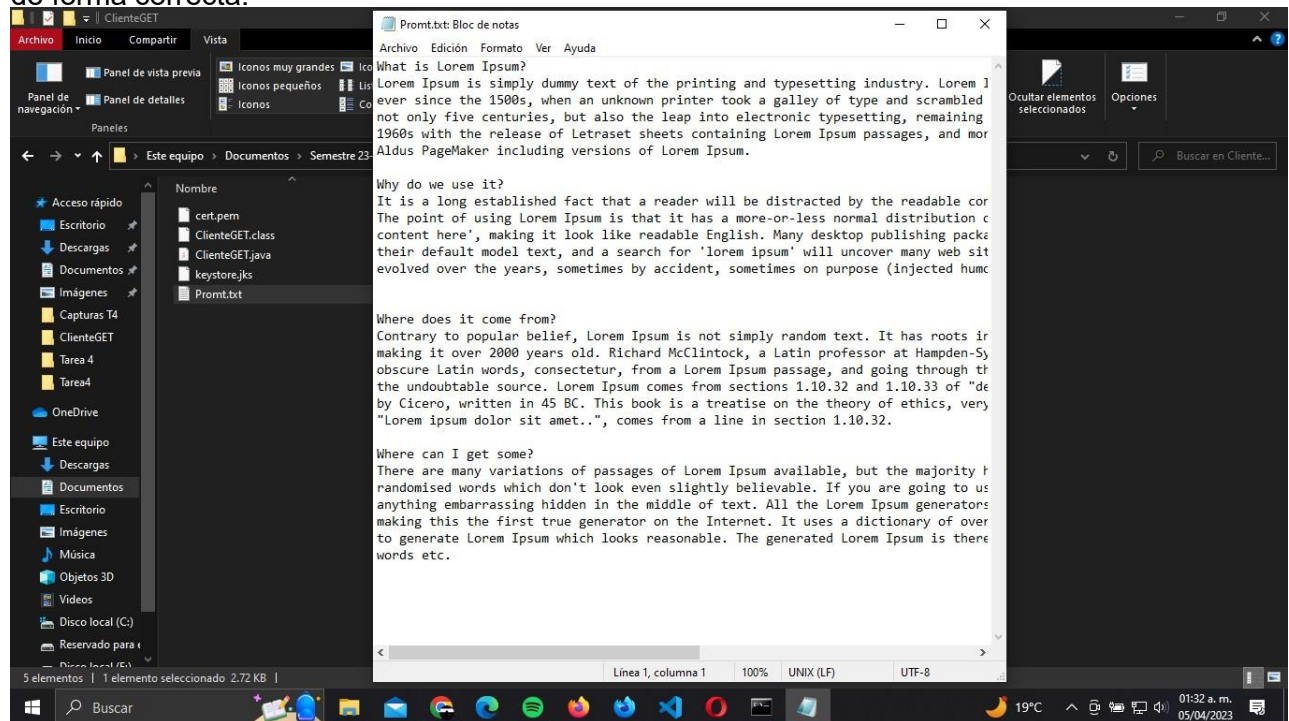


Figura 2.14. Contenido del archivo “Promt.txt” escrito por el ClienteGET.

Finalmente salimos de la máquina virtual.

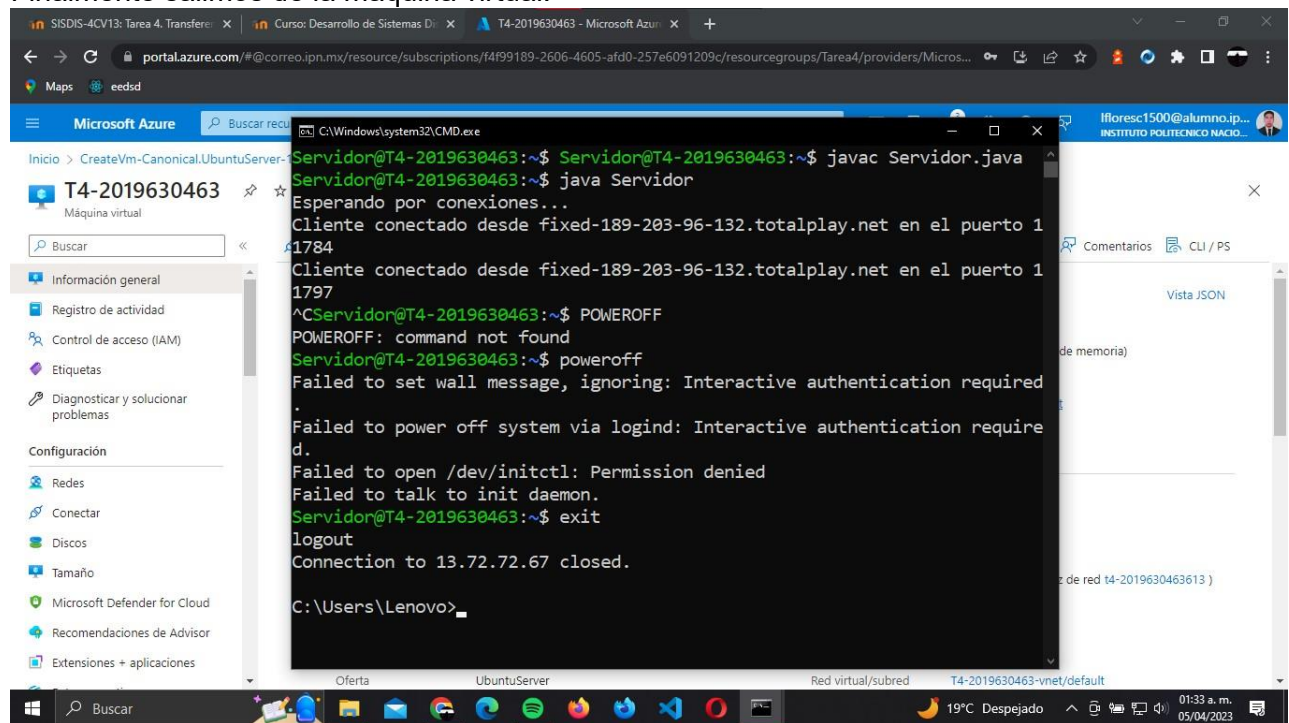


Figura 2.15. Log out de la máquina virtual.

Conversación con Chat GPT.

Luis Antonio Flores Castro.

Hola buena noche, ¿me podrías ayudar realizando lo siguiente por favor? Realizar un programa en Java llamado "Servidor", este servidor debe ser multithread y con sockets. El servidor deberá realizar lo siguiente: Deberá imprimir un mensaje "Esperando por conexiones...". Cuando se reciba una petición GET por parte de algún cliente, se deberá recibir también el nombre de un archivo (sin directorios). El servidor deberá leer el archivo de disco local, si el archivo se puede leer correctamente se deberá enviar al cliente que solicito esta petición el mensaje "OK", la longitud del archivo y el contenido del archivo, en caso de que el archivo no haya podido ser leído se deberá enviar el mensaje "ERROR" al cliente. Cuando se reciba una petición PUT por parte de algún cliente, se deberá recibir también el nombre de un archivo (sin directorios), la longitud del archivo y contenido del archivo. El servidor deberá escribir el archivo en el disco local, si el archivo se pudo escribir de manera correcta en el disco, el servidor enviará al cliente el mensaje "OK", en caso de no haber podido ser escrito se enviará al cliente el mensaje "ERROR". Se deberá utilizar la codificación "Cp850" para que la impresión de letras acentuadas se de forma correcta.

Chat GPT.

¡Hola! Sí, puedo ayudarte a crear un servidor en Java con sockets y multithreading. Aquí te dejo un ejemplo de cómo puedes implementarlo:

```
import java.net.*;
import java.io.*;

public class Servidor {

    private static final int PORT = 8000;

    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = null;

        try {
            serverSocket = new ServerSocket(PORT);
            System.out.println("Esperando por conexiones...");
        } catch (IOException e) {
            System.err.println("Error al crear el socket del servidor: " + e.getMessage());
            System.exit(1);
        }
    }
}
```

```

while (true) {
    try {
        Socket clientSocket = serverSocket.accept();

        System.out.println("Cliente      conectado      desde      "      +
clientSocket.getInetAddress().getHostName() + " en el puerto " + clientSocket.getPort());

        Thread clientThread = new Thread(new ClientHandler(clientSocket));

        clientThread.start();
    } catch (IOException e) {
        System.err.println("Error al aceptar la conexión del cliente: " +
e.getMessage());
    }
}
}

```

```

private static class ClientHandler implements Runnable {

    private final Socket clientSocket;

    public ClientHandler(Socket clientSocket) {
        this.clientSocket = clientSocket;
    }

    @Override
    public void run() {
        try {
            BufferedReader      in      =      new      BufferedReader(new
InputStreamReader(clientSocket.getInputStream(), "Cp850"));

            PrintWriter      out      =      new      PrintWriter(new
OutputStreamWriter(clientSocket.getOutputStream(), "Cp850"), true);

            String inputLine = in.readLine();
            String[] tokens = inputLine.split(" ");

            if (tokens[0].equals("GET")) {
                String fileName = tokens[1];
            }
        }
    }
}

```

```

        File file = new File(fileName);
        if (file.exists() && file.isFile() && file.canRead()) {
            out.println("OK");
            out.println(file.length());
            try (BufferedReader fileReader = new BufferedReader(new
FileReader(file))) {
                String line;
                while ((line = fileReader.readLine()) != null) {
                    out.println(line);
                }
            }
        } else {
            out.println("ERROR");
        }
    } else if (tokens[0].equals("PUT")) {
        String fileName = tokens[1];
        long fileSize = Long.parseLong(tokens[2]);
        File file = new File(fileName);
        try (BufferedWriter fileWriter = new BufferedWriter(new
FileWriter(file))) {
            for (int i = 0; i < fileSize; i++) {
                fileWriter.write(in.read());
            }
            out.println("OK");
        } catch (IOException e) {
            System.err.println("Error al escribir en el archivo " + fileName +
": " + e.getMessage());
            out.println("ERROR");
        }
    } else {
        out.println("ERROR");
    }
}

clientSocket.close();
} catch (IOException e) {

```

```

        System.err.println("Error al manejar la conexión del cliente: " +
e.getMessage());
    }
}
}
}
}
}

```

Este servidor escucha en el puerto 8000 y espera conexiones de clientes. Cuando un cliente se conecta, se crea un hilo para manejar la conexión con ese cliente.

El hilo del cliente recibe la petición del cliente a través de un `BufferedReader` y determina si se trata de una petición GET o PUT. Si es GET, se lee el archivo del disco y se envía su contenido al cliente. Si es PUT, se escribe el archivo en el disco. En ambos casos, se envía un mensaje de confirmación al cliente.

Espero que esto te ayude a crear tu servidor en Java

Luis Antonio Flores Castro.

Ahora, ¿me podrías ayudar con lo siguiente por favor? Realizar un programa en Java llamado "ClientePUT", este ClientePUT deberá ser implementado con sockets, a su vez deberá recibir como parámetros la IP del servidor, el número de puerto y el nombre del archivo a enviar. El ClientePUT deberá realizar lo siguiente: El ClientePUT deberá leer el archivo de disco local, si el archivo se puede leer se conectará al servidor y enviará una petición PUT, el nombre del archivo a enviar, la longitud del archivo y el contenido del archivo. En caso de que el archivo del disco local no haya podido ser leído imprimirá el mensaje "ERROR" seguido del error que se tenga. El ClientePUT deberá esperar la respuesta del servidor, si el servidor responde con el mensaje "OK", el ClientePUT deberá imprimir el mensaje "Archivo fue recibido por el servidor con éxito", en caso de que la respuesta del servidor sea otra se deberá imprimir el mensaje "ERROR: El servidor no pudo escribir el archivo en el disco local". Se deberá utilizar la codificación "Cp850" para que la impresión de letras acentuadas se de forma correcta.

Chat GPT.

¡Por supuesto! Aquí te proporciono el código para el cliente PUT en Java utilizando sockets:

```

import java.io.*;
import java.net.Socket;
import java.nio.charset.Charset;

public class ClientePUT {

    public static void main(String[] args) {

```



```

// Verificar los argumentos
if (args.length != 3) {
    System.err.println("Uso: ClientePUT <IP del servidor> <Número de puerto>
<Nombre del archivo>");
    System.exit(1);
}

// Obtener los argumentos
String servidorIP = args[0];
int puerto = Integer.parseInt(args[1]);
String nombreArchivo = args[2];

// Crear el socket y las variables para los flujos de entrada y salida
Socket socket = null;
DataOutputStream dos = null;
DataInputStream dis = null;
FileInputStream fis = null;

try {
    // Crear el socket
    socket = new Socket(servidorIP, puerto);
    System.out.println("Conexión establecida con el servidor " + servidorIP + ":"
+ puerto);

    // Crear los flujos de entrada y salida de datos
    dos = new DataOutputStream(socket.getOutputStream());
    dis = new DataInputStream(socket.getInputStream());

    // Abrir el archivo y obtener la longitud del archivo
    File archivo = new File(nombreArchivo);
    long longitud = archivo.length();

    // Enviar la petición PUT al servidor
    dos.writeUTF("PUT");

```

```

dos.writeUTF(nombreArchivo);
dos.writeLong(longitud);

// Leer el archivo y enviarlo al servidor
byte[] buffer = new byte[4096];
fis = new FileInputStream(archivo);
int leidos = 0;
while ((leidos = fis.read(buffer)) > 0) {
    dos.write(buffer, 0, leidos);
}
System.out.println("Archivo enviado al servidor");

// Esperar la respuesta del servidor
String respuesta = dis.readUTF();
if (respuesta.equals("OK")) {
    System.out.println("Archivo fue recibido por el servidor con éxito");
} else {
    System.out.println("ERROR: El servidor no pudo escribir el archivo en el
disco local");
}

} catch (IOException e) {
    System.err.println("ERROR: " + e.getMessage());
} finally {
    // Cerrar los flujos y el socket
    try {
        if (fis != null) {
            fis.close();
        }
        if (dis != null) {
            dis.close();
        }
        if (dos != null) {
            dos.close();
        }
    }
}

```

```

        }
        if (socket != null) {
            socket.close();
        }
    } catch (IOException e) {
        System.err.println("ERROR: " + e.getMessage());
    }
}
}
}

```

Este código utiliza los argumentos que se proporcionan en la línea de comandos para conectarse al servidor y enviar el archivo mediante una petición PUT. El archivo se lee en bloques de 4096 bytes y se envía al servidor. Luego, el código espera la respuesta del servidor y muestra un mensaje de éxito o error en función de la respuesta.

Recuerda que debes utilizar la codificación "Cp850" para que la impresión de letras acentuadas se haga de forma correcta.

Luis Antonio Flores Castro.

Ahora, ¿me podrías ayudar con lo siguiente por favor? Realizar un programa en Java llamado "ClienteGET", este ClienteGET deberá ser implementado con sockets, a su vez deberá recibir como parámetros la IP del servidor, el número de puerto y el nombre del archivo a recibir. El ClienteGET deberá conectarse al servidor y enviar la petición GET seguido del nombre del archivo. El ClienteGET deberá esperar la respuesta del servidor, si la respuesta del servidor es un "OK", el ClienteGET deberá escribir el archivo en el disco local, si el archivo se puede escribir se deberá imprimir un mensaje "Archivo ha sido recibido con éxito", en caso de que la respuesta del servidor sea diferente se deberá imprimir el mensaje "ERROR: el archivo no ha sido recibido con éxito". Se deberá utilizar la codificación "Cp850" para que la impresión de letras acentuadas se de forma correcta.

Chat GPT.

Claro, aquí te muestro un posible código para implementar el ClienteGET en Java con sockets:

```

import java.io.*;
import java.net.Socket;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;

public class ClienteGET {
    public static void main(String[] args) {

```

```

// Verificamos que los argumentos necesarios hayan sido proporcionados
if (args.length != 3) {
    System.out.println("Debe proporcionar la dirección IP del servidor, el puerto
y el nombre del archivo");
    return;
}

String serverIP = args[0];
int serverPort = Integer.parseInt(args[1]);
String fileName = args[2];

// Creamos el socket y nos conectamos al servidor
try (Socket socket = new Socket(serverIP, serverPort)) {
    // Creamos los streams de entrada y salida para comunicarnos con el servidor
    OutputStream out = socket.getOutputStream();
    InputStream in = socket.getInputStream();

    // Enviamos la petición GET al servidor seguida del nombre del archivo
    String request = "GET " + fileName + "\n";
    out.write(request.getBytes(Charset.forName("Cp850")));

    // Leemos la respuesta del servidor
    byte[] buffer = new byte[1024];
    int bytesRead = in.read(buffer);
    String response = new String(buffer, 0, bytesRead, StandardCharsets.UTF_8);

    // Verificamos si la respuesta es OK
    if (response.trim().equals("OK")) {
        // Creamos el archivo y escribimos los datos recibidos del servidor
        File file = new File(fileName);
        FileOutputStream fileOut = new FileOutputStream(file);
        int count;
        while ((count = in.read(buffer)) > 0) {
            fileOut.write(buffer, 0, count);
        }
    }
}

```

```

    }

    fileOut.close();

    System.out.println("Archivo ha sido recibido con éxito");
} else {
    System.out.println("ERROR: el archivo no ha sido recibido con éxito");
}
} catch (IOException e) {
    System.out.println("ERROR: " + e.getMessage());
}
}
}

```

Este código se conecta al servidor en la dirección IP y puerto especificados en los argumentos, envía la petición GET con el nombre del archivo y espera la respuesta del servidor. Si la respuesta es "OK", crea un archivo con el nombre especificado y escribe los datos recibidos del servidor en el archivo. Si la respuesta es diferente a "OK", imprime un mensaje de error. La codificación "Cp850" se utiliza al enviar la petición al servidor para que las letras acentuadas se impriman correctamente.

Luis Antonio Flores Castro.

Hola nuevamente, ya he verificado si los programas funcionan correctamente y si lo hacen, excepto el programa "ClientePUT" el ejecutarlo el archivo si es enviado correctamente, pero al esperar la respuesta del servidor parece ser que no la recibe por que aparece un error mencionando algo relacionado a esto ERROR: null, podrías solucionarlo por favor?

Chat GPT.

Sí, claro. El error "ERROR: null" significa que se produjo una excepción y el objeto de excepción es nulo. Esto puede deberse a varias razones, como un problema de red, un problema de servidor o un problema de cliente.

Te sugiero agregar un bloque try-catch en el método enviarArchivo del programa ClientePUT para capturar la excepción y así poder ver el mensaje de error detallado y solucionar el problema. Por ejemplo:

```

import java.io.*;
import java.net.Socket;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;

```



```

public class ClientePUT {
    public static void main(String[] args) {
        // Verificar que los argumentos sean los correctos
        if (args.length != 3) {
            System.err.println("Uso:  java  ClientePUT  <IP  del  servidor>  <puerto>
<archivo>");
            return;
        }

        String ipServidor = args[0];
        int puerto = Integer.parseInt(args[1]);
        String archivo = args[2];

        // Verificar que el archivo exista
        Path archivoPath = Paths.get(archivo);
        if (!Files.exists(archivoPath)) {
            System.err.println("ERROR: El archivo " + archivo + " no existe.");
            return;
        }

        try {
            // Leer contenido del archivo
            byte[] contenidoArchivo = Files.readAllBytes(archivoPath);

            // Crear socket y conectarse al servidor
            Socket socket = new Socket(ipServidor, puerto);

            // Obtener flujos de entrada y salida del socket
            InputStream inputStream = socket.getInputStream();
            OutputStream outputStream = socket.getOutputStream();

            // Enviar petición PUT
            String mensaje = "PUT " + archivo + " " + contenidoArchivo.length + "\r\n";

```

```

        outputStream.write(mensaje.getBytes(Charset.forName("Cp850")));
        outputStream.flush();

        // Enviar contenido del archivo
        outputStream.write(contenidoArchivo);
        outputStream.flush();

        // Leer respuesta del servidor
        BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream,
        Charset.forName("Cp850")));
        String respuesta = reader.readLine();

        if (respuesta == null) {
            System.err.println("ERROR: No se pudo recibir respuesta del servidor.");
        } else if (respuesta.equals("OK")) {
            System.out.println("Archivo fue recibido por el servidor con éxito");
        } else {
            System.err.println("ERROR: El servidor no pudo escribir el archivo en el
disco local.");
        }

        // Cerrar socket
        socket.close();

    } catch (IOException e) {
        System.err.println("ERROR: " + e.getMessage());
    }
}
}

```

Agregué una verificación para asegurarnos de que la respuesta del servidor no sea null. Si la respuesta es null, se imprimirá un mensaje de error indicando que no se pudo recibir respuesta del servidor.

Luis Antonio Flores Castro.

Una pregunta, ¿puedes implementar sockets seguros en java?

Chat GPT.

¡Claro que sí! Java tiene una API llamada Java Secure Socket Extension (JSSE) que permite implementar sockets seguros mediante el protocolo SSL/TLS.

Para utilizar sockets seguros en Java, primero necesitas configurar un certificado SSL/TLS en el servidor y en los clientes. Luego, puedes utilizar las clases de JSSE para establecer conexiones seguras entre el servidor y los clientes.

Luis Antonio Flores Castro.

¿Puedes realizar la implementación de socket seguro para un servidor y un cliente java? solo necesito la implementación de estos sockets seguros, no el programa completo como tal, por favor.

Chat GPT.

Claro, a continuación te muestro un ejemplo de cómo implementar sockets seguros en un servidor y un cliente en Java utilizando SSL/TLS:

Servidor.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import javax.net.ssl.SSLServerSocketFactory;
import javax.net.ssl.SSLSocketFactory;

public class SecureServer {
    public static void main(String[] args) {
        int port = 1234;

        System.setProperty("javax.net.ssl.keyStore", "/path/to/keystore/file");
        System.setProperty("javax.net.ssl.keyStorePassword", "password");
        System.setProperty("javax.net.ssl.trustStore", "/path/to/truststore/file");
        System.setProperty("javax.net.ssl.trustStorePassword", "password");

        try {
```

```

        SSLServerSocketFactory sslServerSocketFactory = (SSLServerSocketFactory)
SSLServerSocketFactory.getDefault();

        ServerSocket serverSocket = sslServerSocketFactory.createServerSocket(port);

        System.out.println("Secure Server started on port " + port);

        while (true) {

            Socket clientSocket = serverSocket.accept();

            System.out.println("Connection established with client " +
clientSocket.getInetAddress().getHostAddress());

            BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));

            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

            String inputLine;
            while ((inputLine = in.readLine()) != null) {

                System.out.println("Received message from client: " + inputLine);

                out.println("Echo: " + inputLine);

            }

            in.close();

            out.close();

            clientSocket.close();

        }
    } catch (IOException e) {

        System.err.println("Error in server: " + e.getMessage());

        e.printStackTrace();

    }

}
}

```

Cliente.

```

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

import java.io.PrintWriter;

```

```

import java.net.Socket;
import javax.net.ssl.SSLSocketFactory;

public class SecureClient {
    public static void main(String[] args) {
        String hostname = "localhost";
        int port = 1234;

        System.setProperty("javax.net.ssl.trustStore", "/path/to/truststore/file");
        System.setProperty("javax.net.ssl.trustStorePassword", "password");

        try {
            SSLSocketFactory sslSocketFactory = (SSLSocketFactory)
            SSLSocketFactory.getDefault();

            Socket socket = sslSocketFactory.createSocket(hostname, port);

            BufferedReader in = new BufferedReader(new
            InputStreamReader(socket.getInputStream()));
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);

            String userInput;
            BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in));

            while ((userInput = stdIn.readLine()) != null) {
                out.println(userInput);
                System.out.println("Received message from server: " + in.readLine());
            }

            in.close();
            out.close();
            stdIn.close();
            socket.close();
        } catch (IOException e) {
            System.err.println("Error in client: " + e.getMessage());
        }
    }
}

```

```

        e.printStackTrace();
    }
}
}

```

Ten en cuenta que para utilizar sockets seguros, necesitarás generar un par de claves (pública y privada) y un certificado. Luego, deberás importar el certificado del servidor en el truststore del cliente y viceversa. Los detalles específicos de cómo generar las claves y certificados y cómo importarlos en Java pueden variar según la implementación de SSL/TLS que estés utilizando.

Luis Antonio Flores Castro.

Al implementar el socket seguro en el servidor se genera el siguiente error: Error al crear el socket del servidor: java.security.NoSuchAlgorithmException: Error constructing implementation (algorithm: Default, provider: SunJSSE, class: sun.security.ssl.SSLContextImpl\$DefaultSSLContext).

creo que el error se puede dar porque no hay un certificado.pem para el servidor, como puede ser implementado?

Chat GPT.

Eso es correcto, para poder utilizar un socket seguro es necesario contar con un certificado que identifique al servidor. Para crear un certificado autofirmado, puedes seguir los siguientes pasos:

1. Genera una clave privada (en este caso, una de 2048 bits):

```
keytool -genkey -alias mydomain -keyalg RSA -keysize 2048 -keystore keystore.jks -validity 3650
```

2. Genera un certificado autofirmado con la clave privada:

```
keytool -export -alias mydomain -keystore keystore.jks -rfc -file cert.pem
```

Una vez que tengas tu certificado autofirmado, puedes utilizarlo en tu servidor para establecer una conexión segura. Para ello, debes modificar las siguientes líneas en el código del servidor:

```
System.setProperty("javax.net.ssl.keyStore", "/path/to/keystore.jks");
```

```
System.setProperty("javax.net.ssl.keyStorePassword", "password");
```

```
System.setProperty("javax.net.ssl.trustStore", "/path/to/keystore.jks");
```

```
System.setProperty("javax.net.ssl.trustStorePassword", "password");
```

Donde /path/to/keystore.jks es la ubicación del archivo del almacén de claves (keystore) que contiene el certificado y la clave privada, y password es la contraseña que se usó para proteger el keystore. Si el keystore y el truststore son los mismos, solo es necesario proporcionar la misma ruta y contraseña en ambas propiedades.

Luis Antonio Flores Castro.

El ClienteGET tiene un error, cuando este programa se ejecuta se le pasa como parámetro una dirección IP, un número de puerto y el nombre del archivo que se quiera escribir, el error esta en que el archivo no se escribe correctamente, es decir, realice una prueba con un archivo txt el cual contenia texto y al ejecutar este programa ClienteGET, no se escribe el archivo de manera correcta, solo se escriben dos números y no la información original, podrías solucionarlo porfavor?

Chat GPT.

Lo siento por la confusión anterior. Parece que el problema radica en que el servidor está enviando los datos en binario y el cliente está intentando escribirlos en el archivo como texto. Para solucionar esto, debemos escribir los datos en el archivo en modo binario. Puedes modificar la sección del código que escribe los datos en el archivo con el siguiente:

```
import java.io.*;
import java.net.Socket;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.util.Scanner;

import javax.net.ssl.SSLSocketFactory;

public class ClienteGET {
    public static void main(String[] args) {
        // Verificamos que los argumentos necesarios hayan sido proporcionados
        if (args.length != 3) {
            System.out.println("Debe proporcionar la dirección IP del servidor, el puerto
y el nombre del archivo");
            return;
        }

        String serverIP = args[0];
        int serverPort = Integer.parseInt(args[1]);
        String fileName = args[2];

        System.setProperty("javax.net.ssl.trustStore",
            "keystore.jks");
        System.setProperty("javax.net.ssl.trustStorePassword", "12345678");
```



```

// Creamos el socket y nos conectamos al servidor
try {
    SSLSocketFactory sslSocketFactory = (SSLSocketFactory)
SSLSocketFactory.getDefault();

    Socket socket = sslSocketFactory.createSocket(serverIP, serverPort);
    // Creamos los streams de entrada y salida para comunicarnos con el servidor
    OutputStream out = socket.getOutputStream();
    InputStream in = socket.getInputStream();

    // Enviamos la petición GET al servidor seguida del nombre del archivo
    String request = "GET " + fileName + "\n";
    out.write(request.getBytes(Charset.forName("Cp850")));

    // Leemos la respuesta del servidor
    byte[] buffer = new byte[8192];
    int bytesRead = in.read(buffer);
    String response = new String(buffer, 0, bytesRead, StandardCharsets.UTF_8);

    // Creamos un Scanner para leer el tamaño esperado del archivo
    Scanner scanner = new Scanner(in);
    String str = scanner.nextLine();
    // Leemos el tamaño esperado del archivo
    long expectedSize = Long.parseLong(str);

    // Verificamos si la respuesta es OK
    if (response.trim().equals("OK")) {
        // Creamos el archivo y escribimos los datos recibidos del servidor
        File file = new File(fileName);
        FileOutputStream fileOut = new FileOutputStream(file);
        int count;
        long totalBytesRead = 0; // Llevamos la cuenta de los bytes leídos
        while (totalBytesRead < expectedSize && (count = in.read(buffer)) > 0) {
            fileOut.write(buffer, 0, count);
        }
    }
}

```

```

        totalBytesRead += count; // Actualizamos la cuenta de bytes leídos
    }
    fileOut.close();

    System.out.println("Archivo recibido exitosamente");
} else {
    System.out.println("ERROR: el archivo no ha sido recibido exitosamente");
}

} catch (IOException e) {
    System.out.println("ERROR: " + e.getMessage());
}

}
}

```

Conclusiones.

A través de la realización de esta tarea pude aplicar las peticiones PUT y GET en java de una mejor forma, ya había trabajado con ellas en formularios usando Javascript y HTML, sin embargo desconocía una aplicación en cuanto a sistemas distribuidos. Al empezar a realizar la tarea decidí hacerla por partes, es decir, primero generar los prompts para servidor y clientes sin implementar sockets seguros ya que me pareció mejor opción primero tener un buen funcionamiento y después implementar los sockets seguros, al generar los 3 programas hubo un problema en el ClientePUT, se producía una excepción por lo cuál le pedí ayuda a ChatGPT para resolverlo, desde mi punto de vista sabía que era un error generado debido a que faltaba incluir la sentencia try catch a un bloque de código, por lo cuál al realizar la modificación con Chat GPT no hubo mayor problema y se resolvió el problema. Después al implementar los sockets seguros tanto en servidor como los clientes, no hubo ningún problema, solo se reemplazaron las líneas de conexión de socket por las de sockets seguros. Cuando ya se realizó la compilación de estos programas se detectó un error el cual indicaba que no encontraba el repositorio de claves y el certificado pem, por lo que le pedí a Chat GPT que me describiera como hacerlo, me dio las instrucciones específicas de que comandos ejecutar en la terminal de mi computador, ya generado el repositorio de claves y el certificado pem, no hubo error al funcionar todos los programas, pero note un detalle, el ClienteGET al realizar la escritura del archivo solo contenía dos números. Interactuando de nuevo con ChatGPT, dedujo que el servidor estaba enviando el archivo como binario, por lo que el ClienteGET debía hacer traducir esto a una codificación para realizar la escritura del texto, realizó la modificación y de esta manera el funcionamiento de todos los programas se dio de manera exitosa.

Finalmente, puedo decir que esta tarea me pareció muy interesante ya que las peticiones PUT y GET las ocupamos todo el tiempo en servicios web y esto me genera un panorama muy amplio de todas las aplicaciones de los sistemas distribuidos. Por otro lado, siento que el uso de Chat GPT se me esta facilitando un poco más ya que estoy generando experiencia en como redactar mis ideas y en saber como funciona más a profundidad.